Discovering and Understanding the Security Flaws of Authentication and Authorization in IoT Cloud APIs for Smart Home

Minglei Guo[1], Zhenghang Xiao[1], Xin Liu[2], and Jianwei Zhuge[1,3(✉)]

[1] Tsinghua University, Beijing, China
{gml20,xzh23}@mails.tsinghua.edu.cn, zhugejw@tsinghua.edu.cn
[2] School of Information Science and Engineering, Lanzhou University, Lanzhou, China
bird@lzu.edu.cn
[3] Zhongguancun Laboratory, Beijing, China

Abstract. The IoT cloud is a vital component of smart homes, responsible for entities' authentication and authorization (A&A). Additionally, the IoT cloud provides many APIs to address complex functional requirements. This paper reports a systematic analysis of A&A security issues in IoT cloud APIs for smart home. To investigate the problem, we first analyze authenticated entities (i.e., devices, users, and families) and identify two categories of flaws based on the existing policies. Next, we introduce a semi-automated tool called *IoTAuthCheck* to discover security flaws in A&A of IoT cloud APIs. *IoTAuthCheck* automatically identifies and replaces credentials in the request, and checks security flaws of the target API by comparing responses before and after the replacement. We conducted experiments using *IoTAuthCheck* on seven popular smart home vendors and found 26 APIs with vulnerabilities that can be classified into six specific types of security flaws. Based on proof-of-concept attacks, we demonstrate that these flaws can cause severe security risks, including sensitive information leakage, malicious data injection, and even unauthorized device control.

Keywords: IoT cloud · Authentication · Authorization · Vulnerability

1 Introduction

Smart homes have brought great convenience to people's daily lives. The smart home architecture comprises three primary components: device, mobile app, and IoT cloud. Through the IoT cloud, users can remotely control their devices using their mobile phones, even when they are not at home. Authentication and

Supported in part by National Natural Science Foundation of China under Grant U1936121.

First Author and Second Author contribute equally to this work.

H. Duan et al. (Eds.): SecureComm 2023, LNICST 567, pp. 205–224, 2025.
https://doi.org/10.1007/978-3-031-64948-6_11

authorization (A&A) are essential responsibilities of the IoT cloud and crucial to the security of the smart home. Security issues of A&A may pose severe security hazards. For instance, suppose an attacker obtains control privileges of a smart lock, it could pose a significant security threat to the home.

Although researchers have conducted security analyses of A&A in the IoT, prior research has yet to conduct a systematic study of IoT cloud APIs. Existing studies typically focus on the security of overall A&A policies [1–4] and do not consider whether various IoT cloud APIs strictly and adequately enforce the policies. Additionally, researchers usually focus on specific process or protocols, such as device binding [5], messaging protocols [6–8], rather than analyzing the security of A&A in IoT cloud APIs. These APIs provide information access, device control, home automation, and many other functions and are mainly based on traditional protocols such as HTTP.

Our study. This paper reports a systematic analysis of A&A security issues in IoT cloud APIs for smart home. The authentication credentials used by cloud APIs can be divided into two categories: *static identifiers* and *dynamic tokens*. Based on the type of authentication credentials used, we divide the IoT cloud APIs into three categories and identify two types of security flaws: weak authentication and weak authorization.

To discover these security flaws automatically, we design *IoTAuthCheck*, a semi-automated tool that helps to check for security flaws of A&A in IoT cloud APIs. Unlike previous tools [9,10] that require a lot of manual operation, such as manually replacing credentials and determining the existence of vulnerabilities, *IoTAuthCheck* checks the necessity of request parameters and combined common naming or format rules to identify credentials. Then, *IoTAuthCheck* replaces the credentials automatically and judges flaws by comparing the responses. Given that most API responses are in JSON format, we combine the JSON similarity calculation algorithm with the comparison of JSON formats to judge whether flaws exist. This approach significantly reduces the manual effort required to identify A&A flaws in IoT cloud APIs.

We conducted experiments using *IoTAuthCheck* on seven popular smart home vendors and identified 26 APIs with security flaws, which we categorized into six specific types based on the involved authentication entities. To demonstrate the risks associated with these flaws, we checked the details of these APIs and performed proof-of-concept attacks, including sensitive data stealing, malicious data injection, and unauthorized device control. We have reported these security flaws to the relevant vendors and received confirmation from five of them. Finally, we provide recommendations to mitigate security threats.

Contributions. We summarize the contributions of this paper as follows:

– *New insights.* We identify two types of security flaws of A&A in IoT cloud APIs and find six specific flaws. Notably, we systematically analyze the authentication entities, among which the *Family* entity have not been discussed in previous research.

- *Semi-automated solutions.* We design and implement *IoTAuthCheck*, a semi-automated tool that identifies security flaws of A&A in IoT cloud APIs for smart home.
- *New findings in practice.* We discovered 26 APIs with security flaws in seven popular vendors using *IoTAuthCheck*. We demonstrate the potential severity of these flaws through three types of proof-of-concept attacks.

2 Background

2.1 Architecture of Smart Home Platforms

IoT devices, IoT apps, and IoT clouds are the three main components of smart home platforms, each with a different role.

- *The IoT device* interacts directly with the physical world to meet users' needs, equipped with various sensors to collect data, receive user commands, and perform related functions.
- *The IoT app* interacts with users and provides a friendly interactive interface that allows users to remotely view device status, obtain environmental data, manage devices to enable smart scenarios.
- *The IoT cloud* acts as a bridge between the app and the device, enabling remote communication. The IoT app sends control messages to the device and receives real-time information about the device through the IoT cloud. During this process, the IoT cloud is responsible for A&A of the user and the device, which is crucial for the security of the smart home platform.

2.2 Authentication in Smart Home

Entities. In the smart home, there are three authenticated and authorized entities: *device*, *user*, and *family (home)*. Users are capable of not only managing their own devices but also establishing families and inviting other family members to join them, thus enabling more efficient device management. Prior work has discussed the A&A of the first two entities but overlooked the *family* entity. The significance of the *family* entity becomes evident in the intricate interrelationships that exist among the three entities, as illustrated in Fig. 1. The concept of a *family* serves as a crucial organizational structure that allows users to group devices and members together, creating a shared environment where collaborative control and management can take place.

Authentication Credential. The IoT cloud utilizes specific credentials for A&A to identify different entities. These credentials can be broadly classified into two categories: *static identifier* and *dynamic token*. The static identifier is typically a unique and invariant identifier, such as a device ID or user ID. The dynamic token is generated by the IoT cloud after the entity undergoes successful authentication. It is variable and usually expires after a certain period.

To make the presentation more clear, we define several notations to identify the authentication credentials of the three entities as below:

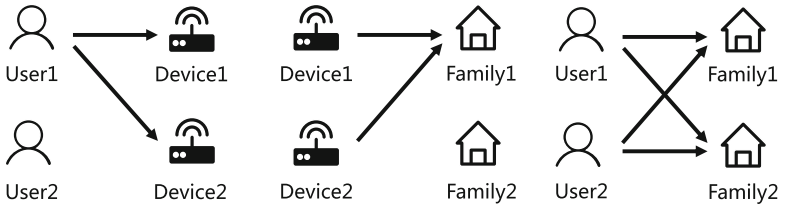

Fig. 1. Interrelationships between three entities in smart home

- For the device, *DevId* is a generic term for the various static identifiers of the device. Some platforms use hard-coded information about the device, such as the MAC address, serial number (SN), etc., while others generate a random ID for each device. *DevToken* typically represents the token issued by the IoT cloud after device binding. If the device token expires or the device's binding changes, the IoT cloud issues a new token for the device.
- For the user, *UserId* represents a unique identifier for each user. *UserToken* represents the token generated by the IoT cloud for the user after the initial user authentication. Common initial user authentication methods include password-based mode, cell phone verification code-based mode, etc. Common token schemes include JSON Web Tokens (JWT), Cookie-Session, etc.
- For the family, *FamilyId* represents a unique identifier for each family. As families are created by users, they typically do not have their own dynamic token for authentication. The IoT cloud typically uses *UserToken* to authenticate families.

In addition to the authentication methods mentioned above, there are public-key-based or certificate-based options available for device authentication. Popular third-party IoT infrastructure providers, such as AWS IoT [11] and Google Cloud IoT [12], employ those more secure solutions. These options will increase the cost of the cloud even though they enable secure device authentication. They are therefore hardly employed by manufacturers of IoT devices [5]. These solutions may also have a detrimental effect on how well devices and the cloud communicate.

3 Preliminaries

3.1 Threat Model

In our threat model, we assume that the adversary has the ability to register accounts and acquire devices within the smart home platform. Additionally, the adversary is capable of intercepting and analyzing communication traffic between their own devices, the cloud, and the app. However, it should be noted that the adversary does not possess the capability to eavesdrop on the communication traffic of other devices and users.

We assume that the attacker can obtain the static identifier of the target's home. We primarily consider currently prevalent device-sharing scenarios where consumers in a hotel or guests in a home may temporarily access IoT devices within the house. A convenient way for the owner to grant access is to invite them to join the home and remove them from it after they leave. During the temporary access, the adversary can collect and analyze the communication traffic to obtain the *DevId*, the *UserId* of the device owner, and the *FamilyId* of the owner's family. After leaving, the adversary can exploit security flaws to perform remote attacks on the target's home. Additionally, since a device is a commodity, an adversary can purchase a device, record the *DevId* and return it to the seller or sell it to someone else.

3.2 Existing Strategies

Before we proceed with the analysis of security flaws in A&A in IoT clouds, it is crucial to have a clear understanding of the strategies implemented by IoT cloud APIs. We categorize these APIs into three categories based on the type of authentication credentials used and analyze their application scenarios separately.

S1: Authenticate with the Static Identifier. Some IoT cloud APIs use the static identifier to authenticate an entity and perform operations on that entity. For example, *DevId* is used by some IoT cloud APIs to authenticate a device and perform operations on that device. Similarly, some APIs authenticate with *UserId* and *FamilyId*, respectively.

S2: Authenticate with the Dynamic Token. Some other IoT cloud APIs utilize the dynamic token to conduct operations on an entity after authenticating it. In this strategy, the dynamic token is used for the entity's authentication and specifies the entity on which the operation is performed. For instance, *UserToken* can be used by a user to retrieve a list of owned devices.

S3: Authenticate with the Dynamic Token and the Static Identifier. Since the three entities - user, device, and family - do not have a one-to-one relationship, some scenarios cannot apply the second strategy. For example, a user has multiple devices and needs to operate on one of them. A common method for APIs is to authenticate the entity with a dynamic token and identify the entity on which the operation is performed with a static identifier, such as *DevId*.

The use of static identifiers in authentication, as in S1, poses a security risk because these identifiers can be easily obtained by adversaries. This causes the weak authentication flaw as attackers can forge the identity of any entity to perform malicious operations. On the other hand, S3 involves the use of dynamic tokens and static identifiers in combination, which can be effective, but it requires

strict authorization rules to be enforced by the IoT cloud. Without these rules in place, the dynamic token will become void, leading to the weak authorization flaw.

4 Methodology

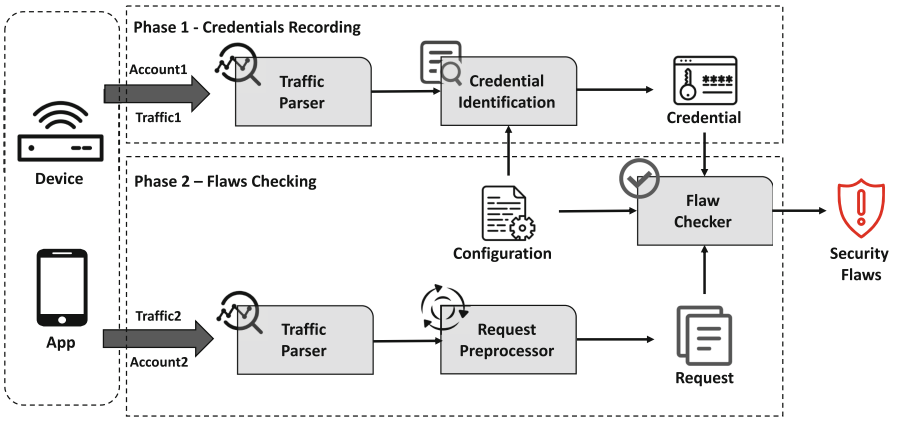

Fig. 2. Structure of *IoTAuthCheck*

In this section, we introduce *IoTAuthCheck*, a semi-automated tool for helping security researchers discover the security flaws of A&A in IoT cloud APIs.

4.1 Overview

The core idea of *IoTAuthCheck* is to identify security flaws in IoT cloud APIs by replacing credentials in the request and comparing the responses before and after the replacement. As shown in Fig. 2, *IoTAuthCheck* is designed as a two-phase approach comprising credentials recording and flaws checking. We will provide a detailed description of these two phases in the following subsections.

4.2 Credentials Recording

This phase aims to record the credentials in the requests sent by apps and devices to the IoT cloud and includes two parts: traffic parsing and credential identification. The former parses each field of the request, and the latter locates and records the credentials.

Traffic Parser. First, *IoTAuthCheck* parses the generated traffic, including the protocol, domain name, method, etc. Since most are HTTP requests, the parsing involves analyzing the header, parameters, and body fields of the requests. Additionally, the parser need to parse at a deep level. For instance, if the value of a parameter is in JSON format, it will continue parsing it.

Credential Identification. After parsing the request, *IoTAuthCheck* identifies the credentials within it. Our approach combines simplicity with efficiency, focusing on minimizing the likelihood of false positives while acknowledging the potential for false negatives.

Our primary strategy is based on some common practices. For example, for static identifiers, most of the parameter naming includes "device" and "id", "device" and "code", "sn", "User" and "id" etc. For dynamic tokens, common names include "authorization," "UserToken," "accessToken," and so on. Parameters in cookies are often used as dynamic tokens. The structure of the parameter value can also be used to determine the type of credential, such as the use of JWT indicating a dynamic token. To address the issue of false positives, *IoTAuthCheck* also provides a user-configuration feature that allows users to manually specify whether certain parameters are credentials. This feature empowers users to define their own rules based on their specific application requirements. For mitigating false negatives, we acknowledge that our rule set, while stringent, may still leave room for improvement. One challenge we face is the absence of ground truth data, which hampers the precise evaluation of false negatives. However, we have designed our tool to strike a balance between simplicity and accuracy.

Because some requests originate from the same API and only differ in a few messages, *IoTAuthCheck* uses a hash function to generate a hash value for each request based on key information such as the protocol, domain name, and path. Then it records the request, hash value, and corresponding credentials. We consider that the requests having the same hash value come from the same API.

4.3 Flaws Checking

This phase aims to identify security flaws of A&A in IoT cloud API corresponding to the request. The phase is composed of three parts: traffic parser, request preprocessor, and flaw checker. The traffic parser operates similarly to the first component of Phase 1. The request preprocessor removes any unnecessary information that may cause interference in the subsequent analysis. Finally, the flaw checker performs flaw checking.

Request Preprocessor. The preprocessor includes two parts: request replaying and parameter cleaning. Repeated requests may result in different responses for APIs involving adding and deleting. For example, a second attempt to delete a resource that has already been deleted may return "Resource does not exist" This can interfere with subsequent determinations after replacing the credentials. Thus, we replay the request and use the response to the replayed request

as the criterion for subsequent judgments. Through practical experimentation, we have observed that, in most cases, replaying a request of specific API does not typically trigger rate limiting. This is primarily because enforcing rate limits for single API could negatively impact the user experience. For example, consumers are less likely to purchase devices that display pop-up warnings after double-clicking a button continuously. Rate limiting mechanisms are more commonly applied to web services or resource-intensive requests.

Moreover, the request can include some pointless parameters that would obstruct our examination. For example, a request carries an *UserToken*, but the cloud does not use it. To address this, we perform parameter cleaning by removing each parameter one by one and comparing the responses.

Check Security Flaws. After preprocessing the request, *IoTAuthCheck* checks for two types of flaws in the API: weak authentication and weak authorization. This involves two steps: checking the credential type and replaying the request after replacing the credentials.

First, *IoTAuthCheck* checks the type of credentials carried in the request. If the request does not carry credentials, it may not require any A&A, for instance, if it's used to get static resources such as images. If the request only carries static identifiers, the corresponding API has the weak authentication flaw. The corresponding API is relatively secure if the request only takes dynamic tokens. However, if the request carries both static identifiers and dynamic tokens, then further investigation is required.

IoTAuthCheck searches for the corresponding credentials in the output of Phase 1 and replaces them before replaying the request. Then it compares the similarity of the responses before and after the replacement. It is important to note that we compare the similarity and not the consistency since some APIs with the same request return different parameters such as message-Id, resulting in a slight difference in the response. Since most APIs return JSON structured data, *IoTAuthCheck* compares the structure of two JSONs and calculates their textual similarity using Deep Distance in the deepdiff [13]. If the structure of the two JSONs is identical, and the text similarity is above a threshold value, we consider that the corresponding API has a weak authorization flaw. Deepdiff compares the differences in every field of JSON format sequentially, rather than just measuring the distance between strings. Consequently, even for fields with relatively small data lengths, deepdiff can accurately detect changes in fields and adjust the distance between two JSON data sets accordingly. This capability ensures precise differentiation of variations, regardless of the size of the changed text within the JSON data.

Furthermore, *IoTAuthCheck* includes a configuration file that allows users to customize the replayed requests to perform checks in complex scenarios. Specifically, users can define a custom function that processes each request before it is sent. This function can be used, for example, to apply signature rules to requests for APIs that sign key fields to avoid replay attacks.

4.4 Implementation

We implement *IoTAuthCheck* using Python based on a popular intercepting proxy tool, mitmproxy, which provides rich APIs that ease our development process. Specifically, we use SQLite to record and store the credentials, and we use the DeepDiff module of the deepdiff package to compare the structure of JSON and calculate the JSON similarity.

5 Experiment

This section presents the experimental phase of our study, where we leveraged *IoTAuthCheck* to identify security vulnerabilities in real-world IoT cloud APIs. We begin by introducing the experimental procedures, followed by an analysis of the results and the detail of each flaw. Finally, we discuss potential ethical risks that may arise from the experiment.

5.1 Experiment Procedures

Experiment Setup. We selected seven popular smart home vendors and purchased their representative IoT devices. These vendors offer a full suite of smart home solutions, including their IoT cloud, IoT devices, and companion apps. As the smart speaker is the home's control center, once it is attacked, it will affect the entire smart home, so we selected the smart speakers from four vendors. Besides, the device types include sweeping robots, purifiers, smart plugs, etc., as shown in Table 1. For each device type, we purchased a pair of devices and assume one device belongs to the victim and the other to the attacker.

In our experiments, the user and the attacker are on different Local Area Networks (LANs). We used two Raspberry Pis to build two LANs for the pair devices to connect to separately. For each vendor, we registered two accounts on the IoT cloud platform, representing the user's and attacker's accounts, and created their own families. We prepared two Android smartphones (Google Pixel 2XL, Android 10), installed companion apps of the devices in both smartphones, and logged in to the apps with two accounts, respectively.

Intercept the Communications. To protect user privacy and improve platform security, smart home platforms usually encrypt communication, which is a challenge for us. Therefore, we overcame this challenge by implementing a Man-in-the-Middle (MITM) attack to intercept the communication. The communication is sourced from manual user operations within the application interface, such as clicking buttons, toggling switches, and setting various parameters.

App-Cloud Communication. This is our main source for collecting APIs. In most cases, these communications are encrypted using TLS, and the app verifies the server's certificate. We rooted the phone and added our certificate to the list of system root certificates. For the app that embeds the server certificate within

Table 1. Experiment Results

Vendors	Device Types	F1			F2		
		F1.1	F1.2	F1.3	F2.1	F2.2	F2.3
Baidu	Smart Speaker	✗	✗	✗	✗	✗	✓
Media	Smart Speaker, Purifier	✗	✗	✗	✓	✓	✗
Himalaya	Smart Speaker	✗	✗	O	✓	O	✗
Haier	Smart Speaker, Sweeping Robot	✓	✓	✗	✓	✓	✗
Ezviz	Smart Plugs, WI-FI Camera	✗	✓	✗	✗	✓	✗
BroadLink	Smart Plugs	✗	✗	✗	✓	✓	✗
Philips Hue & WiZ	Smart Light Bulb	✗	✗	✓	✗	✗	✗

✓: At least one API that has the corresponding flaw.
✗: No APIs that have the corresponding flaw.
O: No relevant APIs.

the app, we utilized Xposed's module and Frida to hook the app's procedure responsible for verifying the certificate.

Device-Cloud Communication. TLS also protects the communication between the device and the cloud, but we cannot directly replace the certificate in the device firmware. We disassembled the device, got the device's shell via the serial port or debug interface. Through this method, we were able to replace the system's certificate, but it was only effective for devices lacking hardware-level protection. We successfully replaced certificates in Haier and Himalaya's smart speakers using this approach. For devices where this method is not possible, we relied on analyzing the communication traffic between the app and the cloud to identify potential vulnerabilities.

Semi-automated Checking. First, we deployed *IoTAuthCheck* on two Raspberry Pis. Then, we logged in with a user's account on the smartphone and performed normal operations such as clicking buttons on the app, controlling the device, etc. During this process, *IoTAuthCheck* automatically identified and recorded authentication credentials for each API. Subsequently, we repeated the same process by logging in with another account on the second smartphone. Finally, *IoTAuthCheck* conducted automated checks and reported the results.

5.2 Experiment Results

We found 26 APIs with security flaws in A&A. All vendors have security issues to a greater or lesser extent. In general, weak authorization flaws are more prevalent. Six vendors have weak authorization flaws, and three have weak authentication flaws. Each category of flaws can be classified into three specific flaws, depending on the entity involved. Table 1 shows the detailed results, including the flaws of each vendor. The details of each specific flaw are as follows:

F1: Weak Authentication. Due to the weak protection of static identifiers in the real world, it is insecure to use only static identifiers for authentication. In addition to the issues of authenticating only by *DevId* discussed in the previous study, such as [1,5,6] etc., we also find the issues of weak authentication in two other entities.

F1.1: Weak Authentication with *DevId*. This flaw may appear in the APIs for acquiring device information, status reporting, and other functions. By exploiting this flaw, attackers can steal sensitive device information, inject false data, etc. We found this issue in Haier Smart Speaker, which poses a serious security risk to the entire smart home platform. Refer to Sect. 6.3 for more details on the attack on Haier Smart Speaker.

F1.2: Weak authentication with *UserId*. We found this flaw in certain cloud APIs of Haier and Ezviz. The flaw is present in APIs that deal with user-related operations, and attackers can exploit it to steal sensitive user data or even modify user configuration data. In addition, this flaw not only affects the security of *user* entity but may also pose a threat to *device* entity. For instance, an API is designed to retrieve all device information owned by a user through *UserId*, which can lead to the leakage of sensitive device information.

F1.3: Weak authentication with *FamilyId*. The authentication of *family* entity is often overlooked in smart home. Some APIs related to *family* entity may only operate through the *FamilyId*, which can compromise the security of all users and devices in the home. In WiZ, an API is designed to obtain all information about users and devices through *FamilyId*, which can result in the leakage of sensitive information in the home.

F2: Weak Authorization Since the *DevToken* is not commonly used with static identifiers, this flaw mainly exists in the authorization between *UserToken* and three types of static identifiers. Although UserToken is used for authenticating users in IoT cloud APIs, it does not verify whether users have the permission to perform operations on the entity. This makes *UserToken*-based authentication meaningless.

F2.1: Weak authorization between *UserToken* and *DevId*. We found this issue in four vendors. This flaw usually exists in the APIs used for users to operate the device through the app, such as device control or status viewing. This flaw directly affects the security of *device*. Besides, we also discovered some strange designs that could impact the security of other entities. For instance, in Midea smart home, a certain API with this flaw is designed to obtain the *FamilyId* of the device's home through *DevId*.

F2.2: Weak authorization between *UserToken* and *FamilyId*. APIs that users use to operate the family may not strictly check if the user has permission to perform operations on the specified family, which could compromise the security of all three entities. For example, attackers could obtain information about users and devices in other families or modify configurations by exploiting this flaw. We found this flaw in several cloud APIs provided by four vendors.

F2.3: Weak authorization between *UserToken* and *UserId*. This flaw is less common, because if only the *user* entity is involved, authentication with only the *UserToken* is sufficient. However, we found instances of this issue in Baidu. The API uses both *UserToken* and *UserId* for authentication, but it does not verify whether the identities of the users identified by the two authentication credentials are consistent, which can result in the leakage of sensitive information of the user identified by the *UserId* and their devices.

5.3 Ethical Considerations and Responsible Disclosure

We experimented very carefully to avoid ethical issues. Specifically, we only tested on our personal accounts and purchased devices, with no impact on other users or devices. Although it is possible to enumerate certain static identifiers through brute-force, we refrained from doing so to avoid excessive cloud API access. Furthermore, our test requests did not contain malicious payloads, so they would not affect the regular operation of cloud services. Additionally, we reported the security flaws we found to the relevant vendors directly or through the cybersecurity hacking competition organizers. We have received confirmation from five vendors, namely Media, Himalaya, Haier, BroadLink, and Philips Hue & WiZ, that they have acknowledged the vulnerabilities and have implemented fixes.

6 Flaw Exploitation

In this section, we aim to investigate the security risks posed by these flaws. Each flaw's security hazard depends on the functionality of API where the flaw exists. We carefully examined the details of the 26 APIs and executed three types of PoC (Proof-of-Concept) attacks: sensitive data stealing, malicious data injection, and unauthorized device control. It should be noted that the only prerequisite for the attack is to obtain the target's static identifier, as described in 3.1. Table 2 shows the results of the attacks on each vendor.

6.1 Sensitive Data Stealing (A1)

Attackers can exploit flaws to steal sensitive information in the home. We found that each flaw can be exploited in this attack. The sensitive information can be divided into privacy data and authentication credentials.

A1.1. Attackers can exploit defects in APIs to steal privacy data from targets, such as home environment data, device status details, configuration data, and other private information. We conducted PoC attacks at Haier, Media, Ezviz, and WiZ. For instance, in Media, we successfully stole the air information in the home (F2.2). While it may be argued that air quality data or other device related information may not directly relate to users' personal privacy, it's important to

Table 2. Attack Results

Vendors	A1		A2	A3
	A1.1	A1.2		
Baidu	✗	✓	✗	✗
Media	✓	✓	✗	✗
Himalaya	✗	✗	✗	✓
Haier	✓	✓	✓	✓
Ezviz	✓	✗	✓	✗
BroadLink	✗	✗	✓	✓
Philips Hue & WiZ	✓	✓	✗	✗

✓: successful attack; ✗: failed attack

recognize that these vulnerabilities can serve as gateways to more invasive data breaches. In WiZ, the app obtains real-time status information on home devices via a WebSocket connection, which only requires the *FamilyId* to complete the three-step process (F1.3). We constructed a script to achieve this process and successfully stole the status information of all devices in the target home by the *FamilyId* of the home. To make matters worse, the *FamilyId* is only six digits long, meaning that attackers can obtain all device state information for any family.

While some of the data mentioned earlier, such as air quality or device status, may not have a direct association with personal privacy, it is crucial to understand that these API vulnerabilities could potentially enable attackers to breach more sensitive information. For instance, hackers could exploit the vulnerabilities to access the video streams from devices like Wi-Fi cameras, posing a more substantial risk to user privacy.

A1.2. In addition to privacy data, attackers can also steal authentication credentials, which we found in Baidu, Haier, Media, and WiZ. As mentioned in the threat model presented in Sect. 3.1, we assume that the adversary can obtain static identifiers, and attackers can exploit this attack to expand the scope and possibility of the attack. For example, the Haier smart speaker obtains the corresponding *FamilyId*, *UserId*, and even *UserToken* through an API located at `/access/ai-access/dot/device/` on `ai.haier.net`. Unfortunately, this API has the weak authentication flaw (F1.1). Attackers can leverage this vulnerability along with other flaws to execute unauthorized device control (see A3 for details).

6.2 Malicious Data Injection (A2)

Attackers can exploit these flaws to inject malicious data into the target and obstruct the user's regular use. As an illustration, an attacker may use the flaw

in the device information reporting API to insert phony smoke alarm data to set off an alert.

We identified this issue in Ezviz, Haier, and BroadLink. In the Ezviz smart home platform, the app modifies some configuration information through an API located at `/v3/family/group/setConf` on `api.ys7.com`. However, this API has the weak authorization flaw (F2.2). By exploiting this API, we successfully prevented the app from pushing device updates of the target home.Similarly, the API used for users to customize instructions on Haier smart speakers has the weak authentication flaw (F1.2), which allows setting custom instructions for the corresponding user only by using *UserId*. As a proof of concept, we added a custom setting to the target speaker, which plays a song instead of providing the weather information when the user ask, "what's the weather like today." Such actions could cause confusion for the user.

6.3 Unauthorized Device Control (A3)

For this attack, the attacker can take control of the device. We discover two ways to exploit the flaw to control the device and complete PoC attacks in Haier, BroadLink, and Himalaya. The details are described below.

Attackers can exploit flaws in APIs designed for device control to launch attacks. We conducted such attacks on BroadLink and Himalaya. The APIs users use to control devices have the weak authorization flaw (F2.1). We successfully controlled another device by collecting traffic controlling our own device and modifying the *DevId*. For Himalaya, this API uses hashes for request signatures to prevent tampering, and we reverse engineered the app to solve this problem. The only prerequisite for carrying out the attack is to obtain the device's *DevId*. In Himalaya, *DevId* is the device's SN, printed on the bottom of the device and easily obtainable by attackers in device-sharing scenarios. In BroadLink, *DevId* is the MAC of the device. In addition to the shared scenario, attackers can even enumerate it to control all the devices.

Attackers can exploit security flaws in the home-scenario function of smart home systems to launch attacks. Currently, users can customize scenes to enhance their experience. As an illustration, a user can configure a scene to turn off a smart plug automatically once the air conditioner is switched off. However, if APIs used to establish scenes have the security flaw, it may permit attackers to generate malevolent scenes in the victim's home and therefore, manipulate the devices.

We conducted a PoC attack on Haier smart speakers, and successfully demonstrated our exploitation in BuTian Cup hacking competition. Our attack combined the vulnerability mentioned in A1.2. It is worth noting that only the smart speaker's *DevId* is required for the attack. Since the smart speaker functions as the control center for the smart home, this allows attackers to gain control of all devices in the home. The attack process is as follows, using the example of turning on the air conditioner. Firstly, we exploited the flaw mentioned in A1.2 to obtain the *FamilyId* of the target home, using the smart speaker's *DevId*. Then, due to the weak authorization flaw in the API located at

`/omssceneapi/scene/v1/user/auto/operation` under `zj.haier.net`, we were able to add a scene to the target home using our own *UserToken*, which turned on the air conditioner after a delay of 5 s. Finally, we waited for 5 s, and the air conditioner turned on. To make matters worse, the *DevId* of the speaker is the MAC address of the device, which means that attacker can carry out large-scale attacks by enumerating *DevIds*.

7 Discussion

In this section, we propose several defensive suggestions to mitigate the security flaws we found, and then discuss some limitations of this work.

7.1 Mitigation

More Secure Policy. Note that this does not refer to the overall policy but emphasizes all APIs' policies. We recommend that IoT clouds adopt User-centered authentication and authorization policies.

- Secure Authentication Policy. For device authentication, the best practice is to use *DevToken* based on the local binding of the user and device. Authentication of other entities should be implemented based on *UserToken*.
- Secure API function design. APIs provided to apps should rely entirely on *UserToken* for functionality. For example, APIs used by users to retrieve information about any entity should be designed to be accessed through *UserToken* rather than static identifiers.

More Comprehensive Authorization Checking. We notice that vendors have recognized the significance of authorization checks but still need to perform them more rigorously and comprehensively. For example, in Ezviz Smart Home, the IoT cloud performed a strict authorization check for the API located at `/v3/sharemng/resource/group/info` under `api.ys7.com`, but not for the API located at `/v3/sharemng/resource/group/member`. Vendors should specify and maintain a clear relationship between the three entities, establish the authorization model, and standardize the implementation policy for all APIs. The best practice is to apply the Role-based Access Control model at the database level and avoid authorization checks at the code level.

Enhance Protection of Identifiers. The prerequisite for attackers to carry out attacks is to obtain static identifiers. Therefore, vendors can mitigate security risks by strengthening the protection of these identifiers. Although not a foolproof solution, this approach can be effective to a certain extent. Firstly, vendors should avoid using weak static identifiers to prevent attackers from easily breaking them through brute force attacks. Instead, they should use long and randomly generated IDs. Additionally, consider using changing identifiers to identify entities, such as the IoT cloud generates a new DevId after the device is re-bound, or updates the DevId periodically.

7.2 Limitation

The Coverage Scope of APIs. We use the communication traffic generated by the device and the app as the data source for testing IoT cloud APIs. Despite our efforts to cover a wide range of functional scenarios, it is not guaranteed to cover all APIs. Additionally, for devices with hardware-level protection, we cannot capture the requests sent from the device to the IoT cloud, and rely solely on requests sent from the app for testing.

Experiment Result. We manually verified the vulnerabilities identified by *IoTAuthCheck* and found no false positives in the experiment result. This is because the criteria we used to confirm the existence of vulnerabilities is relatively strict. However, this approach may result in false negatives in the results, which we cannot confirm since we have no way of knowing the ground truth.

8 Related Work

In this section, we review prior security research about A&A in IoT and discuss how our work is different from previous studies.

Authentication and Authorization in IoT. A&A are critical to the security of IoT, and several studies have conducted security analysis of A&A in IoT, such as [5,14–18]. Yuan et al. [14] performed a systematic study on real-world IoT access delegation, based upon a semi-automatic verification tool they developed. Chen et al. [5] described the life cycle of remote binding with a state-machine model and brought to light questionable practices in the designs of A&A. They conducted security analysis of A&A in specific scenarios, such as cross-cloud [14] and device binding [5], but we focus on various APIs provided by a particular vendor's IoT cloud that involve device control, configuration modification, access to information, and many other scenarios. Furthermore, although Yilian et al. [19] also conducted research on unauthorized access in IoT platforms, their focus was primarily on the detection of authorization vulnerabilities related to device IDs. They did not conduct additional assessments for potential security issues related to other entities or potential authentication-related vulnerabilities associated with these entities. In contrast, our study encompasses a systematic analysis of three entities for authentication and authorization in smart homes. This is particularly significant as prior research has been lacking in relevant investigations pertaining to the *family* entity. Similar to the issues encountered in IoT regarding A&A, online services also face challenges akin to those in IoT cloud platforms. Chaoshun et al. [20] attempted to detect potential authorization issues by substituting credentials in requests with credentials from other accounts. However, we further refine the discussion by categorizing credentials into two types: static identifiers and dynamic tokens. This finer granularity allows us to delve into different authentication challenges.

The exploration of A&A schemes in IoT is a hot topic of research, with related studies, including [21–29]. Roei Schuster et al. [26] designed and implemented a new approach to IoT access control, and the key innovation is to introduce "environmental situation oracles" (ESOs) as first-class objects in the IoT ecosystem. Weijia He et al. [24] suggest that access control should focus on IoT functionality, i.e. the operations that devices can perform, instead of device granularity, to enforce secure IoT access control and identity authentication. However, these solutions have not been widely adopted in practice due to issues related to generality, availability, and other factors. Furthermore, there are few new solutions available for authentication and authorization in IoT clouds.

Security Analysis of Smart Home Platforms. Additionally, researchers have studied the security of smart home platforms, such as [1–4,30–33]. Fernandes et al. [3] discovered the flaws of coarse-grained and sensitive information leakage in SmartThings, and exploited framework design flaws to construct four proof-of-concept attacks. Zhou et al. [1] systematically investigated the complexity of the interactions among the participating entities (i.e., devices, IoT clouds, and mobile apps) and discovered several new vulnerabilities and a spectrum of attacks against real-world smart home platforms. Jia et al. [2] presented the first systematic study on different management channels in IoT, based on a new model-guided approach. They also designed and implemented CGuard, a new access control framework that device manufacturers can easily integrate into their IoT devices to protect end users.

Although these studies discuss security issues about A&A, they all focus on the overall strategy. Our research reveals that even if vendors have designed secure A&A policies, some cloud APIs may not strictly enforce these policies, leading to serious security issues.

Security Analysis of Communication Protocols in IoT. Previous studies have also analyzed the security of communication protocols in IoT, particularly MQTT, which is an OASIS standard messaging protocol for IoT. Related studies include [6–8,34–36]. Jia et al. [6] conducted the first systematic study on the protection of major IoT clouds (e.g., AWS, Microsoft, IBM) put in place for MQTT, and found that the added security measures in the protocol were still vulnerable. Harsha et al. [7] performed a concise study of security inconsistencies in MQTT and its countermeasures. They identified various security loopholes in MQTT. Wang et al. [8] presented MPInspector, a automatic and systematic solution for vetting the security of MP implementations. They evaluated MPInspector on three popular Messaging Protocols, including MQTT, CoAP and AMQP, implemented on nine leading IoT platforms, and identifies 252 property violations.

While above work investigate the A&A issues of the messaging protocol, our study focus on various IoT cloud APIs, mostly based on traditional communication protocols like HTTP and WebSocket. Due to the differences in protocols, their A&A policies are quite different.

9 Conclusion

This paper reports the first comprehensive investigation of authentication and authorization security issues in IoT cloud APIs for smart home. To gain insight into these security issues, we analyze the existing strategies and identify two categories of security flaws. Additionally, we design a semi-automated tool for checking the security flaws of A&A in IoT cloud APIs and conducted experiments on seven popular smart home vendors. Through real-world case studies, we reveal the insecure design and practice of A&A in IoT cloud APIs, which could result in sensitive information leakage, malicious data injection, and even unauthorized device control.

References

1. Zhou, W., et al.: 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), pp. 1133–1150 (2019)
2. Jia, Y., et al.: Who's in control? On security risks of disjointed IoT device management channels. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 1289–1305 (2021)
3. Fernandes, E., Jung, J., Prakash, A.: Security analysis of emerging smart home applications. In: IEEE Symposium on Security and Privacy (SP), pp. 636–654. IEEE 2016 (2016)
4. Liu, H., Li, C., Jin, X., Li, J., Zhang, Y., Gu, D.: Smart solution, poor protection: An empirical study of security and privacy issues in developing and deploying smart home devices. In: Proceedings of the 2017 Workshop on Internet of Things Security and Privacy, pp. 13–18 (2017)
5. Chen, J., et al.: 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, pp. 222–233 (2019)
6. Jia, Y., et al.: Burglars' IoT paradise: understanding and mitigating security risks of general messaging protocols on IoT clouds. In: IEEE Symposium on Security and Privacy (SP). IEEE 2020, pp. 465–481 (2020)
7. Harsha, M., Bhavani, B., Kundhavai, K.: Analysis of vulnerabilities in MQTT security using Shodan API and implementation of its countermeasures via authentication and ACLs. In: 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 2244–2250. IEEE (2018)
8. Wang, Q., et al.: Mpinspector: a systematic and automatic approach for evaluating the security of IoT messaging protocols. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 4205–4222 (2021)
9. Authz. https://github.com/PortSwigger/authz. Accessed April 2023
10. Auth Analyzer. https://github.com/PortSwigger/auth-analyzer. Accessed April 2023
11. "AWS IoT Authentication - AWS IoT Core". https://docs.aws.amazon.com/iot/latest/developerguide/client-authentication.html. Accessed April 2023
12. "Google Cloud IoT Authentication". https://cloud.google.com/iot/docs/concepts/device-security. Accessed April 2023
13. "Deep Distance". https://zepworks.com/deepdiff/current/deep_distance.html. Accessed April 2023

14. Yuan, B., Jia, Y., Xing, L., Zhao, D., Wang, X., Zhang, Y.: Shattered chain of trust: understanding security risks in cross-cloud IoT access delegation. In: 29th USENIX Security Symposium (USENIX Security 20), pp. 1183–1200 (2020)
15. Jin, Z., Xing, L., Fang, Y., Jia, Y., Yuan, B., Liu, Q.: P-verifier: understanding and mitigating security risks in cloud-based IoT access policies. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 1647–1661 (2022)
16. Zhang, Y., Ma, S., Li, J., Gu, D., Bertino, E.: Kingfisher: unveiling insecurely used credentials in IoT-to-mobile communications. In: 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 488–500. IEEE (2022)
17. Yu, S., Zhang, X., Huang, P., Guo, L., Cheng, L., Wang, K.: Authctc: defending against waveform emulation attack in heterogeneous IoT environments. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, pp. 20–32 (2020)
18. Oh, S.-R., Kim, Y.-G.: Security requirements analysis for the IoT. In: 2017 International Conference on Platform Technology and Service (PlatCon), pp. 1–6. IEEE (2017)
19. Li, Y., Yang, Y., Yu, X., Yang, T., Dong, L., Wang, W.: IoT-apiscanner: detecting API unauthorized access vulnerabilities of IoT platform. In: 2020 29th International Conference on Computer Communications and Networks (ICCCN), pp. 1–5. IEEE (2020)
20. Zuo, C., Zhao, Q., Lin, Z.: Authscope: towards automatic discovery of vulnerable authorizations in online services. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 799–813 (2017)
21. Tian, Y., et al.: Smartauth: user-centered authorization for the internet of things. In: 26th USENIX Security Symposium (USENIX Security 17), pp. 361–378 (2017)
22. Jia, Y.J., et al.: Contexlot: towards providing contextual integrity to appified IoT platforms. In: NDSS, vol. 2, no. 2. San Diego, p. 2 (2017). S. Unviersity
23. Fernandes, E., Paupore, J., Rahmati, A., Simionato, D., Conti, M., Prakash, A.: FlowFence: practical data protection for emerging IoT application frameworks. In: 25th USENIX Security Symposium (USENIX Security 16), pp. 531–548 (2016)
24. He, W., et al.: Rethinking access control and authentication for the home internet of things (IoT). In: 27th USENIX Security Symposium (USENIX Security 18), pp. 255–272 (2018)
25. Chi, H., Zeng, Q., Du, X., Luo, L.: Pfirewall: semantics-aware customizable data flow control for home automation systems, arXiv preprint arXiv:1910.07987, 2019
26. Schuster, R., Shmatikov, V., Tromer, E.: Situational access control in the internet of things. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1056–1073 (2018)
27. Ongun, T., Oprea, A., Nita-Rotaru, C., Christodorescu, M., Salajegheh, N.: The house that knows you: user authentication based on IoT data. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 2255–2257 (2018)
28. Rashid, M.A., Pajooh, H.H.: A security framework for IoT authentication and authorization based on blockchain technology. In: 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). IEEE 2019, pp. 264–271 (2019)

29. Fang, H., Qi, A., Wang, X.: Fast authentication and progressive authorization in large-scale IoT: how to leverage AI for security enhancement. IEEE Netw. **34**(3), 24–29 (2020)
30. Fernandes, E., Rahmati, A., Jung, J., Prakash, A.: Decentralized action integrity for trigger-action IoT platforms. In: Proceedings 2018 Network and Distributed System Security Symposium (2018)
31. Sikder, A.K., Babun, L., Aksu, H., Uluagac, A.S.: Aegis: a context-aware security framework for smart home systems. In: Proceedings of the 35th Annual Computer Security Applications Conference, pp. 28–41 (2019)
32. Yu, J.-Y., Kim, Y.-G.: Analysis of IoT platform security: a survey. In: 2019 International Conference on Platform Technology and Service (PlatCon), pp. 1–5. IEEE (2019)
33. Ding, W., Hu, H., Cheng, L.: IoTSafe: enforcing safety and security policy with real IoT physical interaction discovery. In: The 28th Network and Distributed System Security Symposium (NDSS 2021) (2021)
34. Kim, J.Y., Holz, R., Hu, W., Jha, S.: Automated analysis of secure internet of things protocols. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 238–249 (2017)
35. Chen, F., Huo, Y., Zhu, J., Fan, D.: A review on the study on MQTT security challenge. In: 2020 IEEE International Conference on Smart Cloud (SmartCloud), pp. 128–133. IEEE (2020)
36. Bhawiyuga, A., Data, M., Warda, A.: Architectural design of token based authentication of MQTT protocol in constrained IoT device. In: 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA), pp. 1–4. IEEE (2017)