International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Leakage of Authorization-Data in IOT Device Sharing: New Attacks and Countermeasure

Mr. Ghayatadak Jaibhimrao Eknath
Student of Dept. of Computer Engineering.
Vishwabharati Academy's college of Engineering, Ahmednagar, Maharashtra, 414201, India.
jaighayatadak14@gmail.com

***Abstract*—Device sharing among users is a common function- ality in today's IoT clouds. Supporting device sharing are the delegation methods proposed by different IoT clouds, which we find are heterogeneous and ad-hoc — IoT clouds use various data (e.g., device ID, product ID, and access token) as authorization certificates. In this paper, we report the first systematic study on how the authorization-data are managed in IoT device shar- ing. Our study brought to light the security risks in today's IoT authorization-data management, identifying 6 authorization-data leakage flaws. To mitigate such flaws, we propose an approach to hide the authorization-data from the delegatee (a.k.a., the user authorized to access the devices) without disrupting the device sharing services. We propose *SecHARE*, an automated tool to patch the vulnerable IoT clouds. We applied *SecHARE* to 3 popular open-source IoT clouds. Results have shown the compatibility, effectiveness, and efficiency of *SecHARE*. We have made *SecHARE* publicly available.**

***Index Terms*—Cyber-physical systems, IoT security, authori- zation-data protection.**

I. I NTRODUCTION

TODAY'S IoT (Internet of Things) cloud platforms are providing more and more functionalities to meet the users' various requirements. Device sharing among multiple users is one of the most popular functionalities supported by the mainstream IoT clouds (e.g., AWS IoT [1], Samsung SmartThings [2],
Philips Hue [3] and MiHome [4]). Device sharing allows the owner/admin-user to delegate the access right to the device to other users and clouds (which we call the delegatee). Prominent
examples include that the owner of a Philips Hue device inviting other Philips users to control her device (by issuing a whitelistID for the delegatee user [5]) and the owner of a SmartThings smart home authorizing Google Home cloud to control her device (by sending the device ID of the SmartThings device and an OAuth token [6] to Google Home cloud). Serving this purpose are the delegation mechanisms proposed by different IoT vendors, which we found are heterogeneous and ad-hoc. In specific,different IoT clouds use different types of data (e.g., device ID, product ID, OAuth token) as the authorization certificates (which we call the authorization-data, see Section III).
Also, different types of data are with different changeability, for example, the device ID (set when the device is created) is usually unchangeable, while access tokens are usually changeable (e.g., updated by the owner). Previous researches have revealed that vulnerabilities in these IoT delegation mechanisms could expose users to security and safety risks [5]. However, little have done to systematically study how the authorization-data are managed (e.g., creation, distribution, and deletion) in the real-world IoT clouds.
Risks in poor authorization-data management: Permission issues have always been one of the key concerns of IoT security, and access control community has also been studying delegation
of authority issues, finding privacy leaks, incomplete credential revocation, overprivileged authorization, and incorrect policy enforcement [5], [7], [8], [9]. However, in today's IoT cloud
ecosystem, access control is not only distributed but also heterogeneous and ad-hoc, so authorization-data protection remains an open problem. Fernandes et al. [9] found that the vulnerable event management in SmartThings enables the attackers to obtain device identifiers to send fake fire alarms. Bin et al. [5] found that the insecure cross-cloud IoT delegation could also result in the leakage of device information and OAuth token, leading to unauthorized access to the victim devices. Given the severe consequences of authorization-data leakage (e.g., privacy leakage and safety threats), it is emerging to understand and mitigate this problem. To secure the authorization in IoT, Fernandes et al. [10] presented DTAP to prevent an untrusted trigger-action platform from misusing compromised OAuth tokens, while Andersen

TABLE I
FLAWS DISCOVERED IN POPULAR IOT PLATFORMS

	Flaw 1			Flaw 2			Flaw 3			Flaw 4			Flaw 5			Flaw 6		
	FD[1]	PB[2]	DoS[3]	FD	PB	DoS	FD	PB	DoS	FD	PB	DoS	FD	PB	DoS	FD	PB	DoS
Kaa_E*	✓	✓								✓	✓							
Kaa_O*																✓	✓	
ThingsBoard				✓	✓	✓				✓	✓							
ThingsKit				✓	✓													
JetLinks							✓	✓	✓				✓	✓				
ThingsPanel													✓	✓				

* Kaa_E is the enterprise version of Kaa. Kaa_O is the open-source version of Kaa.
[1] Falsified Data. [2] Privacy Breaches. [3] Denial of Service attacks.

et al. [11] presented WAVE, an authorization framework supporting decentralized trust and transitive fine-grained delegation/revocation. However, these approaches require significant
changes to the existing IoT cloud platforms and devices. Real-world adoption and deployment of DTAP [10] or WAVE [11] may take a long time, until all compatibility and usability issues
are resolved. A timely solution that is lightweight, compatible with existing IoT clouds, and effective at securing the authorization-data is needed. Findings and impacts: In this paper, we report the first sys-
tematical study on how the authorization-data are managed in today's IoT device sharing. Specifically, we studied 6 popular IoT clouds to investigate the life-cycle of the device data,
Especially the authorization-related device data, which we call authorization-data for short.
Our study shows that, in the absence of security standards/guidance, today's IoT clouds usually develop their homegrown mechanisms to support device sharing, resulting in heterogeneous and ad-hoc authorization-data management. In specific, we find IoT clouds use various types of data with different changeability as authorization-data (see Section III). Moreover, our study shows that, due to the lack of understanding on the security implications of the authorization-data, today's
IoT clouds often adopt vulnerable authorization-data management mechanisms. We have identified 6 authorization-data leakage flaws in the evaluated IoT clouds (see Section IV). Leveraging these flaws, attackers can use the leaked authorization-data to emulate the victim devices for device state and event forgery attacks (e.g.a fake alarm event), privacy theft attacks (e.g., inferring the absence/presence of the victim via obtaining the state of the victim's devices) and deny of service (DoS) attacks (e.g., disconnecting a sub-device). As shown in Table I, we summarize the severe consequences of these attacks as falsified data (FD), privacy breach (PB) and deny of service (DoS). Moreover, we found the flaws identified could expose a large number of IoT users and other IoT clouds, as well as organizations, and vendors in various fields, to security risks (Section VI-A). We report all flaws to the relevant parties and have received 5 CNVDs [12] by the time we write this paper. Defense with shadow authorization-data: To mitigate such flaws, we propose a method that can hide the actual authorization-data from the delegatee to avoid leakage with-out interrupting the device sharing services. Specifically, when delegating access right to the delegatee, we generate a new copy of authorization-data (e.g., device ID') that is different from the actual authorization-data (e.g., device ID), and record the mapping relationship between them (e.g., device ID
→ device ID'). We call the new copy of authorization-data the shadow authorization-data. Then, we send the shadow authorization-data (device ID') to the delegatee. Upon receiving the delegatee's request to access the delegated device, we transfer the shadow authorization-data back to the actual authorization-data according to the recorded mapping relationship (device ID' → device ID), and then perform authorization check based on the actual authorization-data (device ID).
To revoke the delegatee's access right, we delete the shadow authorization-data and the mapping relationship between the shadow and actual authorization-data. Such a workflow can avoid leaking the actual authorization-data to the delegatee users. Moreover, the malicious delegatee users will not be able use the shadow authorization-data to gain unauthorized access to the victim devices after his access right is revoked. In addition, the whole process is made transparent to the users — the delegator and delegatee users can use the device sharing services as usual as they already do in today's IoT systems. Automated patching: Further, we design and implement SecHARE, a tool that automatically patches the vulnerable IoT clouds. Specifically, SecHARE takes as input a configuration file that specifies the names of the sensitive methods operating the authorization-data (e.g., the assignDevice-ToCustomer() function in ThingsBoard [13] used to share
a device to a delegatee user), automatically identifies such methods and inserts necessary operations into the bytecodes for security enhancement (see Section V). Moreover, SecHARE implements an automatic configuration file generator to reduce the manual efforts needed to specify the configuration files (see Section V-D). We applied SecHARE to 3 popular open-source IoT clouds, ThingsBoard [13], Kaa [14] and JetLinks [15]. Our evaluation
shows that SecHARE can effectively mitigate the authorization- data leakage flaws with negligible/acceptable overheads and can be easily deployed into today's IoT ecosystem. We have made SecHARE publicly available [16].
Contributions: We summarize our contributions as follows:

• New understanding: We performed the first systematic study on how the authorization-data are managed in the IoT clouds, which reveals the security-critical weaknesses in today's IoT authorization-data management.
• New findings: We investigated 6 popular IoT clouds and identified 6 authorization-data leakage flaws, which expose many IoT devices/users to realistic security risks with severe consequences.
• New techniques: We proposed a new method to mitigate the flaws in IoT authorization-data management, and developed/released support for automated securing IoT authorization data. We implemented our proposed method and demonstrated its usability, efficiency, and compatibility with existing IoT cloud systems. The insights and techniques of our study can help secure not only today's but also future IoT authorization-data management.

II. BACKGROUND

A. IoT Cloud Architecture and Its Device Sharing Device control and its automation: A typical IoT architecture
include the IoT devices, the cloud and the user console (e.g.,mobile app or web app). To control
an IoT device, as shown in Fig. 1, the device owner first register her device to the cloud, with the device bound to her user account. To access the device, the owner initiates a request through her user console. Upon receiving the request, the cloud performs an authorization

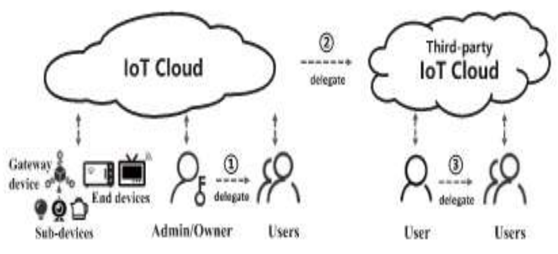


Fig. 1. IoT cloud architecture and its device sharing.

check on the user and send the command to the target device if the check passes. Moreover, users can define automation rules (a.k.a., trigger-action rules) for automated device control —
when receiving the trigger of the rule, the cloud automatically performs the action. For example, a user can set an automation rule to work in that if the motion sensor detects movement, turn
on the light. Device types: As shown in Fig. 1, there are three types of IoT devices:
(1) The end device, devices that connect to the IoT cloud directly and do not manage sub-devices;
(2) The gateway device, a gateway device connects to the IoT cloud directly and manages/connects to other sub-devices;
(3) The sub-devices, a sub-device is managed by and connected to a gateway device. Note that, the sub-devices usually connect to the gateway device through protocols like Zigbee [17], Z-wave [18] and BLE [19], while the gateway device acts as an agent for the sub-devices to
communicate with the IoT cloud. A sub-device usually can be added to and removed from the gateway device under the user's operation.
IoT messaging protocol: Many IoT messaging protocols (e.g., MQTT [20], HTTP [21], CoAP [22], LwM2M [23] and AMQP [24]) are used by today's IoT clouds, among which MQTT is the most widely used [25]. MQTT adopts a publish–
subscribe messaging pattern [26]. For two clients to communicate with each other, the MQTT server (called the MQTT broker) uses topics to define the message classes; the message receiver (called the subscriber) subscribes to the topics to show its interest
in these message classes; the message sender (called the publisher) publishes messages to specific topic(s); upon receiving the published message, the broker identifies the corresponding
topic and transmits the message to all its subscribers. To secure the messaging process, before accepting/delivering messages from/to the clients, the MQTT broker usually performs authentication and authorization check based on the Username, Password or topic contained in the messages. Note that, different implementations of MQTT broker might customize such security checks — performing the checks based on other information (e.g., customized tokens) or even performing no check at all. Device sharing in IoT clouds: Device sharing among multiple users is commonly supported by today's IoT clouds, which enables the admin-user/owner of the devices to delegate access right to the devices to others.
• Device sharing within a single IoT cloud: The owner can share her devices with other users under the same cloud (① and ③ in Fig. 1). For example, IoT clouds including HomeKit [27],
MiHome [4] and SmartThings [2] enables the owner to invite other users to join the owner's smart home system and delegate access rights to the devices to the invited users.
• Cross-cloud device sharing: Cross-cloud delegation [5] is also commonly seen in today's IoT clouds for owners to share the devices with users from third-party clouds (② in Fig. 1).
Cross-cloud device sharing is usually implemented with OAuthprotocol — the delegator cloud issues an OAuth

token to the
delegatee cloud. In real world, the delegator clouds are usually the clouds maintained by the device manufacturers, such as Philips Hue [3], SmartThings [2] and MiHome [4], while the
delegatee clouds are the third-party cloud services providers like Google Home [28] and IFTTT [29]. B. Aspect-Oriented Programming
Aspect-oriented programming (AOP) was proposed by Gregor et al. [30], which aims to address the cross-cutting problem during effective application modularization. It can add additional behaviors non-invasively without changing the original design/code. “Weaving” is one of the terms in AOP, which is the process of applying the functionality that needs to be
extended to the target object. “Weaving” can be divided into four different types based on the time when the behavior is woven into the target class: (1) Compile-Time Weaving, which weaves the behavior at the source code compilation with a special compiler; (2) Post-Compile/Binary Weaving which weaves the behaviors into the compiled file with a special compiler; (3) Load-Time Weaving which uses a special class-loader to weave the behavior when the class is loaded into the Java virtual machine;
(4) Run-Time Weaving which weaves the behavior during the execution of the program. The two most popular frameworks of AOP are AspectJ [31] and Spring AOP [32]. AspectJ supports the former three weaving types, while Spring AOP supports the latter two types. In this paper, we leveraged AspectJ to insert security-enhancement behaviors into the vulnerable IoT clouds with Load-Time Weaving (see Section V-C).
III. LIFE -CYCLE OF AUTHORIZATION-DATA

To investigate how the authorization-data is managed in today’s IoT device sharing, we studied over 10 mainstream IoT clouds, including AWS IoT [1], Alibaba Cloud IoT [33],
Google Home [28], Tuya [34], SmartThings [2], IFTTT [29], ThingsBoard [35], Kaa [14], [36], JetLinks [37], ThingsKit [38] and ThingsPanel [39]. Definition of authorization-data: To enable the delegatee user to access the delegated devices, the IoT cloud usually would send certain data to the delegatee user. Such data would then be used for authorization check when the delegatee user attempts to access the devices. We call the data transmitted to the delegatee during device sharing and used for authorization checks during the delegatee’s access to the device the authorization-data.Type of authorization-data: Recall that, in the absence of the standard/guidance on how to securely share devices, the implementation of device sharing by different IoT clouds are heterogeneous. Specifically, various types of authorization-data are used in today’s IoT clouds, including public available information (e.g., app-version-name), identifiers (e.g., device ID and product ID), access tokens, MQTT topics, MQTT passwords and HTTP/CoAP URLs. Changeability of authorization-data: Further, some types of authorization-data (e.g., device ID and product ID) are determined by the IoT clouds and are unchangeable by the users, while other types of authorization-data are changeable under the operations from the authorized users (including both the device owner and the delegatee users). For example, the “endpoint token” of a device under the Kaa Enterprise cloud [36] can be changed by users via revoking the old token and activating a new one. Life-cycle of authorization-data: Moreover, we find that the authorization-data could be created, accessed, updated, trans- mitted, deleted or deactivated in different phases of the device management. We summarize the life-cycle of authorization-data as follows.

• Add device: The life-cycle of authorization-data starts with adding the device. Usually, when the owner registers/binds a new device, the IoT cloud determines/generates the unchangable information for the device, which are used as unchangable authorization-data by some IoT cloud. For example, the device ID and application version is used as unchangable authorization- data by SmartThings [2] and Kaa Enterprise [36] respectively.

• Share device: When the owner shares a device to the delegatee user, the IoT clouds may generate new authorization-data (e.g., issuing a new access token) or reuse the ex- isting data (e.g., device ID) as authorization-data. Then, the IoT clouds would transmit the authorization-data to the delegatee.

• Unshare device: After the owner revokes the access right from the delegatee user, the authorization-data would be removed from the delegatee user’s system (e.g., her mobile app).Moreover, the IoT clouds could deactivate/update some of the changeable authorization-data, such as the OAuth token.

• Query device information: An authorized user could query the cloud for the device information. The responses to such queries might contain authorization-data. For example, the ThingsBoard [35] transmits the authorization-data of access token to the querying user.

• Update device data: The authorized users are usually allowed to update the device data, including both the basic device data (e.g., device name) and the changeable authorization-data (e.g., access token).

• Access device: When a user attempts to access a device, the user usually sends to the cloud an access request that carries authorization-data, which is accessed and verified by the cloud for authorization check. used for authorization check when the delegatee user attempts to access the devices. We call the data transmitted to the delegatee during device sharing and used for authorization checks during the delegatee's access to the device the authorization-data. Type of authorization-data: Recall that, in the absence of the standard/guidance on how to securely share devices, the implementation of device sharing by different IoT clouds are heterogeneous. Specifically, various types of authorization-data are used in today's IoT

clouds, including public available information (e.g., app-version-name), identifiers (e.g., device ID and product ID), access tokens, MQTT topics, MQTT passwords and HTTP/CoAP URLs.

Changeability of authorization-data: Further, some types of authorization-data (e.g., device ID and product ID) are determined by the IoT clouds and are unchangeable by the users, while other types of authorization-data are changeable under the operations from the authorized users (including both the device owner and the delegatee users). For example, the "endpoint token" of a device under the Kaa Enterprise cloud [36] can be changed by users via revoking the old token and activating a new one. Life-cycle of authorization-data: Moreover, we find that the authorization-data could be created, accessed, updated, transmitted, deleted or deactivated in different phases of the device management. We summarize the life-cycle of authorization-data as follows.

• Add device: The life-cycle of authorization-data starts with adding the device. Usually, when the owner registers/binds a new device, the IoT cloud determines/generates the unchangable information for the device, which are used as unchangable authorization-data by some IoT cloud. For example, the device ID and application version is used as unchangable authorization-data by SmartThings [2] and Kaa Enterprise [36] respectively.

• Share device: When the owner shares a device to the delegatee user, the IoT clouds may generate new authorization-data (e.g., issuing a new access token) or reuse the existing data (e.g., device ID) as authorization-data. Then, the IoT clouds would transmit the authorization-data to the delegatee.

• Unshare device: After the owner revokes the access right from the delegatee user, the authorization-data would be removed from the delegatee user's system (e.g., her mobile app).Moreover, the IoT clouds could deactivate/update some of the changeable authorization-data, such as the OAuth token.

• Query device information: An authorized user could query the cloud for the device information. The responses to such queries might contain authorization-data. For example, the ThingsBoard [35] transmits the authorization-data of access token to the querying user.

• Update device data: The authorized users are usually allowed to update the device data, including both the basic device data (e.g., device name) and the changeable authorization-data (e.g., access token).

• Access device: When a user attempts to access a device, the user usually sends to the cloud an access request that carries authorization-data, which is accessed and verified by the cloud for authorization check.

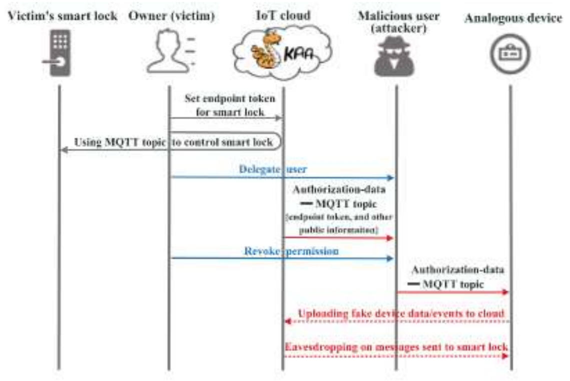

Victim's smart lock
Owner (victim)
IoT cloud
KAA
Malicious user (attacker)
Analogous device
Set endpoint token for smart lock
Using MQTT topic to control smart lock
Delegate user
Authorization-data — MQTT topic (endpoint token, and other public information)
Revoke permission
Authorization-data — MQTT topic
Uploading fake device data/events to cloud
Eavesdropping on messages sent to smart lock

Fig. 2. Flaw 1 — MQTT topic leakage in Kaa Enterprise.

Flaw 1. MQTT topic leakage: Kaa Enterprise [36] is an IoT cloud platform that supports device

sharing among users and supports multiple messaging protocols including MQTT, CoAP and HTTP. When a device connects to the Kaa Enterprise cloud using MQTT protocol, the MQTT topic is used as authorization-data. That is, Kaa Enterprise authorizes the device based on the topic contained in the messages (see Section II-A) — only the devices that could provide valid topics are allowed to publish messages to the Kaa Enterprise cloud. To construct a valid topic, Kaa Enterprise requires the owner to set the "endpoint token" (a string) for the device when adding the device. The endpoint token is regarded as a secrecy that can only be accessed/updated by authorized users (including the owner and the delegatee user). With the endpoint token, Kaa Enterprise constructs the topic for the device by adding other publicly available information (e.g., the application version) to the endpoint token. Hence, a typical MQTT topic in Kaa Enterprise could be kp1/{application_version}/dcx/{endpoint_token}/json. However, we found the the authorization-data of Kaa Enterprise cloud (e.g., the MQTT topic) can be obtained by the delegatee user. Recall that, the delegatee user is allowed to update the endpoint token of the delegated device. Therefore, the delegatee user can gain a valid endpoint token by updating it. Using the updated valid token and other public information, the delegatee user can obtain a valid copy of MQTT topic. Note that, updates to endpoint tokens are automatically handled and synced by the cloud and do NOT notify the device owner or affect the owner's use of the device. Moreover, the delegatee user's updates to the endpoint tokens remain valid even afterhis access right is revoked, resulting in that, the MQTT topic leaked to the delegatee remain valid and unchanged after the revocation. Therefore, a malicious delegatee user could leverage the leaked MQTT topic to send/receive unauthorized messages to/from the Kaa Enterprise cloud after his access right is revoked. PoC exploit on Flaw 1: To exploit Flaw 1, we used our test account and a MQTT-enabled smart lock (a virtual device) to implement an end-to-end PoC attack. Specifically, as shown in Fig. 2, the owner first set the endpoint token of the smart lock on the Kaa Enterprise cloud platform for the victim smart lock to communicate with the cloud. Then, the owner shared the smart lock with the attacker. The attacker then updated the smart lock's endpoint token and obtained a valid MQTT topic. After that, we let the owner revoked the attacker's access right. Then, we wrote a program using Python (publicly available at [16]) to pretend to be the victim smart lock to communicate with the Kaa Enterprise cloud. As shown in Table I, we were able to send forged messages (FD) and receive unauthorized messages (PB) to/from the cloud. Flaw 2. MQTT username leakage: In addition to the MQTT topic leakage (see Flaw 1), we found that the MQTT username is also used as authorization-data and could be leaked to the attackers in ThingsBoard [35] and ThingsKit [38]. ThingsBoard is a popular open-source IoT platforms that supports MQTT, HTTP, CoAP and LwM2M. When a MQTT-enabled device communicates with the ThingsBoard cloud, the device is required to provide a valid MQTT username, which is used for MQTT's authentication/authorization check (see Section II-A). Therefore, a valid and unique MQTT username will be assigned to the device when the owner adds the device to ThingsBoard. Specifically, during the device adding, the owner or the cloud would set an "access token" for the new device. The access token then is used as the MQTT username for the device to communicate with the ThingsBoard cloud. The problem we identified here is that the access token is accessible to the delegatee user — when the delegatee user queries the ThingsBoard cloud for the information about adevice delegated to him, ThingsBoard would send information that contains the device's access token to the delegatee user. Moreover, the access token does NOT change or become invalid after the owner revokes the access right from the delegatee user. Consequently, a delegatee user with malicious intentions could obtain the MQTT username (a.k.a., the access token) when he is authorized and reuse the leaked MQTT username to stealthily communicate with the cloud after he loses access right to the device.PoC exploit on Flaw 2: We conducted a PoC attack to ex- ploit the Flaw 2 in ThingsBoard. In specific, we configured a virtual gateway device connected to other sub-devices in the ThingsBoard cloud (Fig. 3). We then temporarily shared the gateway device to the attacker. At this point, the attacker can obtain the access token from the ThingsBoard cloud to form the authorization-data (e.g., MQTT username). After we revoked the attacker's permission, we tried to use our attacking programs ([16]) to communicate with the ThingsBoard cloud. We found that, with the leaked MQTT username, we were able to send forged messages and receive messages to/from the ThingsBoard cloud. In addition, we were able to publish messages to the v1/gateway/disconnect topic to disconnect the sub-devices under the victim gateway from the cloud (DoS in Table I). Note that, We found the exact same problem discussed above in the ThingsKit [38] platform. An attacker can leverage Flaw 2 to send falsified data and receive private information of the victim to/from the ThingsKit cloud (see Table I). Flaw 3. MQTT password leakage: JetLinks [37] is another open-source IoT cloud platform that supports MQTT protocol and uses the MQTT password as authorization-data [45] for devices to communicate with the cloud. Specifically, when adding a new device, the owner needs to set a “secureKey” and a “secureId” for the device. To communicate with the cloud, the device sends MQTT packages with the MQTT password set to md5(secureID+“|”+timestamp+“|”+secureKey) and another field filled with the timestamp in plaintext. Upon receiving such a package from the device, JetLinks checks the correctness and freshness of the MQTT password using the timestamp received and its own copy of secureId and secureKey. Only the devices that pass such checks are allowed to communicate with the cloud. The problem in JetLinks cloud is similar to that in ThingsBoard (Flaw 2). The authorization-data (e.g., the secureId and secureKey) is obtainable to the delegatee user and does NOT change after the owner revokes the delegatee user’s access rights. Therefore, a malicious delegatee user can use the leaked authorization-data to communicate with the cloud even after his access right is revoked. PoC exploit on Flaw 3: Exploiting Flaw 3 is also similar to the exploitation of Flaw 2, as shown in Fig. 4. The key challenge was for the attacker to obtain the secureId and secureKey. This was done by capturing the traffic between the attacker’s user console (the web-based application provide by JetLinks) and the JetLinks cloud. In our PoC attack, the attacker successfully extracted the secureId and secureKey from the packets/traffic sent from the JetLinks cloud to the user console. Then, with our PoC attacking programs [16], the attacker was able to conduct all the three attacks (e.g., FD, PB, and DoS in Table I).

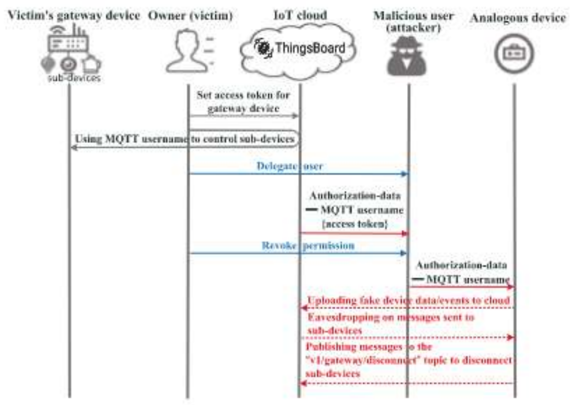

Victim's gateway device
sub-devices
Owner (victim)
IoT cloud
ThingsBoard
Malicious user (attacker)
Analogous device
Set access token for gateway device
Using MQTT username to control sub-devices
Delegate user
Authorization-data — MQTT username (access token)
Revoke permission
Authorization-data — MQTT username
Uploading fake device data/events to cloud
Eavesdropping on messages sent to sub-devices
Publishing messages to the "v1/gateway/disconnect" topic to disconnect sub-devices

Fig. 3. Flaw 2 — MQTT username leakage in ThingsBoard.

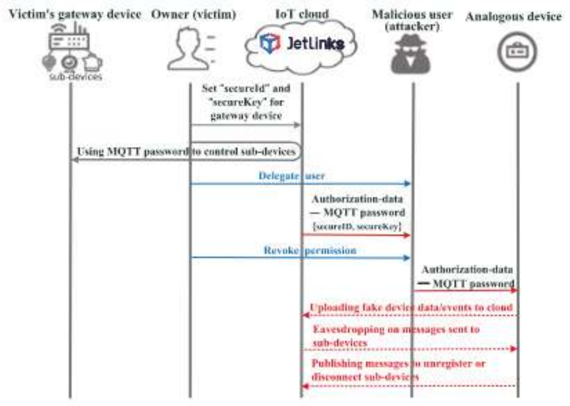

Victim's gateway device
sub-devices
Owner (victim)
IoT cloud
JetLinks
Malicious user (attacker)
Analogous device
Set "secureId" and "secureKey" for gateway device
Using MQTT password to control sub-devices
Delegate user
Authorization-data — MQTT password {secureID, secureKey}
Revoke permission
Authorization-data — MQTT password
Uploading fake device data/events to cloud
Eavesdropping on messages sent to sub-devices
Publishing messages to unregister or disconnect sub-devices

Fig. 4. Flaw 3 — MQTT password leakage in JetLinks.

Flaw 4. URL leakage: Recall that, ThingsBoard supports HTTP for the devices to communicate with the cloud. We identified the problem of URL leakage in the HTTP messaging of the ThingsBoard. Specifically, in ThingsBoard’s HTTP messaging, the URL (e.g., http(s):// host:port/ api/ v1/ access_token/telemetry) is used as the device’s authorization-data and is unique for each device. Anyone who knows the URL can communicate with the cloud on behalf of (or pretend to be) the device. The problem here is that the URL could also be leaked to the
attacker, who then could use the URL to communicate with the ThingsBoard cloud maliciously.To make matters worse, even an attacker who has never been authorized to access the device before can obtain the URL and
conduct the attacks (see Table I). For example,the attacker could monitor all the traffic in the victim’s home WiFi network to extract the URL. PoC exploit on Flaw 4: The PoC exploitation of Flaw 4 is rather straightforward. As outlined in Fig. 5, we let the victim owner shared the virtual smart bulb to the attacker. The attacker was able to extract the URL from the traffic between his user console and the ThingsBoard cloud. After the owner revokes
the attacker’s access right, we found the attacker was

able to communicate with the ThingsBoard cloud using our PoC attack programs [16]. Note that, we found the same problem (Flaw 4) in the Kaa Enterprise platform (see Fig. 5), which also supports HTTP messaging. We omit the detailed discussion for simplicity.Flaw 5. Device identifier leakage: We found that the device identifier is used as authorization-data in JetLinks's HTTP messaging and ThingsPanel's MQTT messaging, both of which are vulnerable.

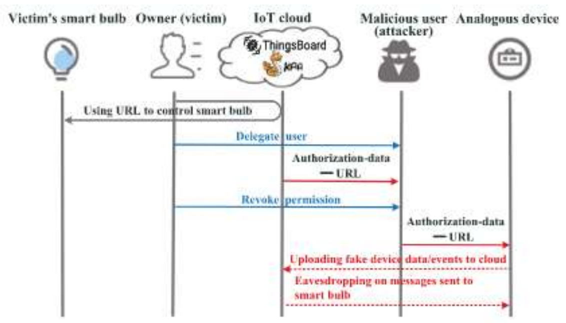


Fig. 5. Flaw 4 — URL leakage in ThingsBoard and Kaa Enterprise.

In JetLinks's HTTP messaging, JetLinks exposes a public URL (http://server-address/report-property) for the devices to communicate with the cloud (Fig. 6). To authenticate and authorize a device, JetLinks requires the device to provide a valid device ID (which is created when the device is added to the cloud and is unchangeable) in the packets sent to the URL.However, such an unchangeable authorization-data (e.g., device ID) is accessible to the delegatee users (by querying the device data from the cloud), which leads to the FD and PB attacks in JetLinks as shown in Table I.

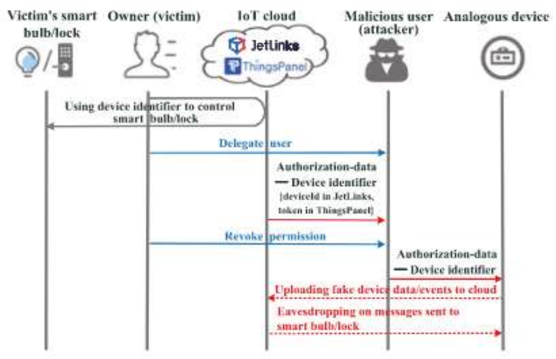

Victim's smart bulb/lock
Owner (victim)
IoT cloud
JetLinks
ThingsPanel
Malicious user (attacker)
Analogous device
Using device identifier to control smart bulb/lock
Delegate user
Authorization-data — Device identifier (deviceId in JetLinks, token in ThingsPanel)
Revoke permission
Authorization-data — Device identifier
Uploading fake device data/events to cloud
Eavesdropping on messages sent to smart bulb/lock

Fig. 6. Flaw 5 — Device identifier leakage in JetLinks and ThingsPanel.

Moreover, in ThingsPanel's MQTT messaging, ThingsPanel uses the same MQTT topic for all the devices and requires each device to provide its unique identifier (e.g., the token set by the owner when adding the device) for authorization check. However, as shown in Fig. 6, when a device is shared to the delegatee user, the delegatee user can obtain the device's token in message push log of the device. Such data leakage could lead to the FD and PB attacks in ThingsPanel as shown in Table I. PoC exploit on Flaw 5: We confirmed Flaw 5 in both JetLinks and ThingsPanel with our PoC attacking programs [16]. Flaw 6. SDK token leakage: Kaa open-source [14] is a open-source IoT cloud platform that supports flexible device definition and creation. Specifically, Kaa open-source provides the owners an endpoint SDK (a library that exposes many useful APIs for the device to use) for them to create devices with various functionalities. Each time a device is created, the cloud would generate a unique token (which we call the SDK token) for the device and store the SDK token into the device's own copy of SDK. The SDK token is then used for the cloud to perform authorization check when a device attempts to communicate with the cloud (Fig. 7). Moreover, when the owner authorizes a delegatee user to access a device, the delegatee user is allowed to download the SDK of the delegated device. As a result,the delegatee user can further obtain the device's SDK token from the downloaded SDK. Besides, the SDK token does NOT change when the owner revokes the delegatee user's access right. Therefore, a malicious delegatee user can leverage this flaw to stealthily communicate with the cloud, resulting in FD and PB attacks (see Table I). PoC exploit on Flaw 6: In our PoC attack, as outlined in Fig. 7, the owner used the Kaa open-source SDK to create a virtual smart lock (whose SDK token is set as 2wXVH-wXD6TR_cAdr5RoWal6K0Q by the cloud). Then, the owner delegated the smart lock to the attacker. The attacker downloaded the smart lock's SDK and wrote an attacking program [16] that used the SDK along with the SDK token in it to connect to the cloud. We found that the attacking program can still successfully communicate with the Kaa open-source cloud after the attacker's permission was revoked.

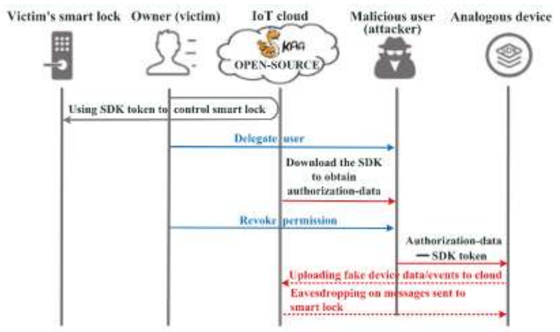

Victim's smart lock
Owner (victim)
IoT cloud
Kaa
OPEN-SOURCE
Malicious user (attacker)
Analogous device
Using SDK token to control smart lock
Delegate user
Download the SDK to obtain authorization-data
Revoke permission
Authorization-data — SDK token
Uploading fake device data/events to cloud
Eavesdropping on messages sent to smart lock

Fig. 7. Flaw 6 — SDK token leakage in Kaa open-source.

TABLE II
RECEIVED CNVD IDS

IoT cloud platform	CNVD ID
`ThingsBoard` (Flaw 2, Flaw 4)	*CNVD-2022-27848*
`JetLinks` (Flaw 3, Flaw 5)	*CNVD-2022-38276*
`Kaa Enterprise` (Flaw 1)	*CNVD-2022-41097*
`ThingsKit` (Flaw 2)	*CNVD-2022-56053*
`ThingsPanel` (Flaw 5)	*CNVD-2022-84690*
`Kaa open-source` (Flaw 6)	*CNVD-2023-38442*

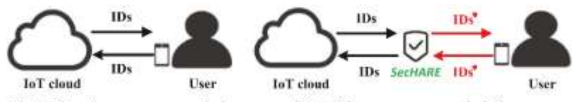
Responsible disclosure: We report all flaws to relevant parties, who all acknowledged the seriousness of the problems. We have received 6 CNVDs [12] (see Table II). Ethical consideration: The PoC attacks are conducted using our own accounts/devices in our testing environment, without disrupting the real-world IoT services or users.

V. SYSTEM DESIGN AND IMPLEMENTATION

In this section, we elaborate on the design and implementation of SecHARE, an automated tool to patch the vulnerable IoT clouds for authorization-data protection, which can be easily applied to today's IoT clouds. We have made SecHARE publicly available [16].

A. Overview

At a high level, the IoT clouds should ensure that the authorization-data transmitted in device sharing will not be leaked to attackers, preventing the unauthorized access to the devices from the attackers. To fix the authorization-data leakage flaws (discussed in Section IV), we propose a usability preserving protection method that replaces the actual authorization-data with the shadow authorization-data and transmits the shadow authorization-data to the delegatee user without interrupting the device sharing services — the owner and the delegatee users can use the IoT services as normal as they already do in today's IoT systems. The security enhancement is achieved by hiding the actual authorization-data from the delegatee users. In specific, as illustrated in Fig. 8(a), without our protection, the actual authorization-data (e.g., IDs) is transmitted to the delegatee user during device sharing, which could lead to the problems discussed in Section IV. In contrast, SecHARE works as a proxy during authorization-data transmission: 1) when the cloud sends authorization-data (e.g, ID) to the delegatee user, SecHARE generates a shadow copy of the authorization-data (e.g., ID') and send it to the user; 2) when a message from the user arrives
at the cloud, SecHARE converts the shadow authorization-data to the actual authorization-data and the inner process logic of the cloud uses the actual authorization-data for further processing. Notably, the shadow authorization-data is generated using the same format of that of the actual authorization-data (e.g., a 20-bit string). As a result, impacts on the normal functionalities introduced by SecHARE can be minimized. To this end, we developed SecHARE to automatically patch the vulnerable codes of the IoT clouds.



(a) Without our proposed defense (b) With our proposed defense

Fig. 8. Authorization-data transmission between the cloud and the users (with and without our proposed defense).

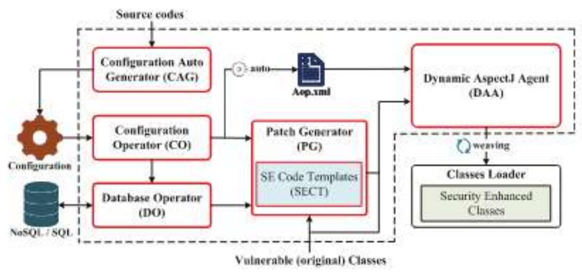

Source codes
Configuration Auto Generator (CAG)
auto
Aop.xml
Dynamic AspectJ Agent (DAA)
Configuration
Configuration Operator (CO)
Patch Generator (PG)
SE Code Templates (SECT)
weaving
Classes Loader
Security Enhanced Classes
NoSQL / SQL
Database Operator (DO)
Vulnerable (original) Classes

Fig. 9. Architecture of *SecHARE*.

Architecture: Since different IoT clouds use different types of authorization-data and define different methods/functions to create, access, update, transmit, delete and deactivate the uthorization-data. We need a method to automatically identify the methods/functions that operate the authorization-data and patch these methods/functions to fix possible authorization-data leakage in a way that does not impact the usage of IoT services. To this end, as shown in Fig. 9, we built SecHARE, which is
composed of 5 components: a Configuration Operator (CO), a Database Operator (DO), a Patch Generator (PG), a Dynamic AspectJ Agent (DAA), and a Configuration Automatic Generator (CAG). Essentially, SecHARE generates patches for the vulnerable cloud with the predefined Security Enhancement Code Templates (SECT) based on our usability preserving defense (see Section V-B) and leverages the AspectJ [31] (an AOP framework, see Section II-B) framework to insert these patches into the IoT cloud when the classes are loaded into the Java virtual machine. Specifically, to apply SecHARE to patch an IoT cloud, we need to deploy and execute SecHARE along with the IoT cloud. Then, as shown in Fig. 10, CO takes as input the configuration file (which specifies the methods/functions operating the authorization-data) to generate the Aop.xml file for the DAA to use1 . CO also outputs information (e.g., the specified authorization-data to protect and the names of methods

need to be patched) to the PG. Along with the database operation APIs provided by DO, the PG then generates the patch codes. Taking as input the Aop.xml and the patch codes generated by PG, the DAA leverage the AspectJ framework to compile the patching codes and weave the additional/security-enhancement behaviors (defined by the patch codes) into the IoT cloud's original vulnerable classes at loading time, allowing the IoT cloud to use Security Enhanced Classes to manage/operate the authorization-data.

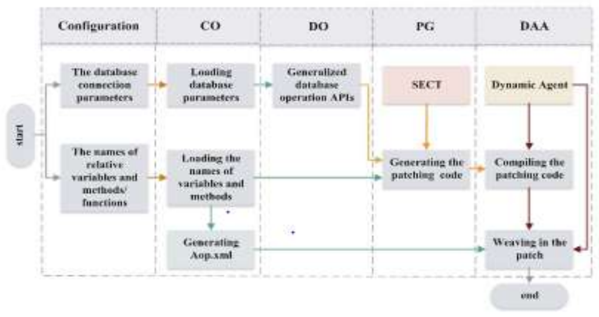

Configuration
CO
DO
PG
DAA
start
The database connection parameters
The names of relative variables and methods/ functions
Loading database parameters
Loading the names of variables and methods
Generating Aop.xml
Generalized database operation APIs
SECT
Generating the patching code
Dynamic Agent
Compiling the patching code
Weaving in the patch
end

Fig. 10. Workflow of *SecHARE*.

B. Usability Preserving Defense

IoT device sharing is vital to today's IoT cloud. Almost all IoT clouds support such functionality, for users widely require it (e.g., sharing devices to family members, Airbnb guests, babysitter, etc.). Therefore, the key to authorization-data leakage solution is how to avoid disrupting the normal IoT device sharing
service. Our solution is to provide a usability preserving defense that is made transparent to the users — they can use the device sharing services as normal as they already do. Specifically, our proposed defense leverages a simple yet effective data mapping scheme to prevent authorization-data leakage. In specific, after the owner shares her device to a delegatee user, the IoT cloud needs to transmit the authorization-data to the delegatee user. Instead of transmitting the authorization-data directly to the delegatee user (as today's IoT clouds do), we generate a shadow copy of authorization-data, record the mapping relationship between the actual authorization-data and the shadow authorization-data and then transmit the shadow
authorization-data to the delegatee user. The delegatee user then uses the shadow authorization-data to access the delegated device. Upon receiving the access request from the delegatee user, the cloud extracts the shadow authorization-data from the request, transfers the shadow authorization-data to the actual authorization-data based on the mapping records stored by the cloud, and uses the actual authorization-data for authorization check. When the owner revokes the delegatee user's access right, the cloud delete the shadow authorization-data and its corresponding mapping record. Hence, even if the shadow authorization-data is leaked to and preserved by the malicious delegatee users, he will not be able to leverage the shadow authorization-data to gain unauthorized access to the device. Note that, all the operations (e.g., data-mapping, data-storage and data-deletion) are performed automatically by the backend cloud, which are transparent to the users. Therefore, we could fix
the authorization-data leakage problems in today's IoT clouds while preserving their usability. Example: Taking Flaw 2 (Section IV) as an example, Fig. 11 illustrates how our defense operates the authorization-data and shadow authorization-data. Recall that, ThingsBoard uses the MQTT Username as authorization-data in its MQTT messaging. Therefore, to share the device to the delegatee user, ThingsBoard generate the shadow authorization-data (MQTT Username') for the actual authorization-data (MQTT User-name). Then, the MQTT Username' is transmitted to the delegatee user, instead of MQTT Username. When the delegatee user is authorized, he can use MQTT Username' to access the device normally. After the delegatee user's permission is revoked, ThingsBoard removes the MQTT Username'. As a result, the delegatee user can no longer access the device, even if he preserved the MQTT Username' when he was authorized.
Discussion: Recall that, today's IoT clouds use both changeable and unchangeable authorization-data. When the changeable authorization-data is leaked to the attacker, the owner might help to mitigate the problem by changing/updating the authorization-data each time he revokes access right from a delegatee user. However, this approach relying on users to ensure the security may not be ideal. First, real world owners might not be aware of the problem or forget to update the authorization-data. Second, the device might be shared to multiple delegatee users. Updating the authorization-data when revoking one of the delegatee users might cause the other delegatee users cannot access the device,either. If the unchangeable authorization-data is leaked to the attacker, the owner has little to do to ease the problem since she cannot update the authorization-data.

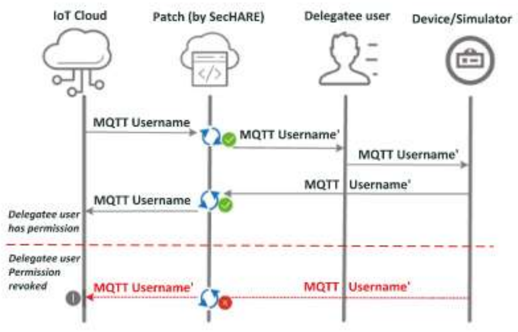


Fig. 11. Usability preserving defense to Flaw 2.

C. Automated Patching

How to adapt to different IoT clouds' implementations of device sharing and keep the performance overhead to minimal are vital to automated patching.Adaptability and scalability: The implementations of device sharing in today's IoT clouds are heterogeneous — defining multiple methods/functions to operate the various types of authorization-data. Hence, it is particularly important for the patch scheme to adapt to most (if not all) of the IoT clouds and even scale to new IoT clouds. For better adaptability and scalability, we consider the follow aspects.

• Configuration guided patching: Based on our understanding on the lifecycle of authorization-data (see Section III), we cannot generate a single unified patch for all of the clouds. Instead, we leverage a configuration that specifies the
methods/functions operating authorization-data defined in a specific IoT cloud to generate the unique patch for the cloud. Note that, our patch scheme is general and scalable. To patch another
IoT cloud, we simply ask for a new configuration file and patch the cloud accordingly. We further developed CAG to reduce the manual efforts for specifying the configuration file (see Section V-D).

• Minimal changes to the system. It is also essential to ensure easy deployment and minimal changes to existing systems. To this end, we adopt the AOP (see Section II-B) technique to only weave security-enhancement behaviors into the original system without breaking the overall workflow/logic design. Moreover, the weaving is done automatically by our tool at the loading time of the classes, requiring minimal (or no) manual intervention from the IoT cloud manager/developer. • Supporting SQL/NoSQL database: Our scheme stores the mapping relationships between the authorization-data and
shadow authorization-data in the database. Also, such data are stored in concordance with the data managed within the cloud platform, and are inaccessible to users. Consider the usage of different types of databases, we develop DO to provide universal APIs for database access and implement DO to support both SQL and NoSQL databases. Minimal performance overhead: Low end-to-end latency is important in IoT device control. To minimize the latency overhead, we only introduce additional computation to the cloud-side while the client-side (the device and user console) remains unchanged. Since the clouds are usually with strong computing capabilities, the overhead should be negligible (see Section VI-B).

TABLE III
COMMON WORDS/AFFIXES AT SOME PHASES OF THE LIFE-CYCLE OF AUTHORIZATION-DATA

Phase[1]	Common Words[2]
Add Device	*add, save, create, regist-*
Delete Device/User	*del-, remove, cancel*
Share Device	*auth-, right, permission, assign, claim, grant, deploy, delegate*
Unshare Device	*auth-, right, permission, revoke, unassign, disclaim, undeploy*

D. Automatic Generation of Configuration Files

Essentially, the configuration file specifies the implementation details of device sharing, including which data/variables are used as authorization-data and which meth- ods/functions operate the authorization-data. We expect the users of SecHARE (e.g., a developer/manager of the IoT cloud) to provide the configuration file, for they would already
know the implementation details. Nevertheless, we develop CAG to help the users to specify the configuration file, reducing the manual efforts needed to use our tool. CAG mainly focuses on automatically identify the names of methods/functions that operate the authorization-data. Note that, it is possible for CAG to identify a non-related method/function
as method/function that operates the authorization-data. Hence, we let CAG list all the methods/functions it identified and let the
user to delete or add methods/functions from/to the list. Specifically, we investigated 50 IoT cloud projects on Github to learn the naming pattern/habit of the IoT programming. We found that the methods/functions defined in the 8 different phases of the authorization-data's lifecycle (see Section III) can be divided into two categories: (1) The methods/functions that
have a common naming pattern, including Add device, Delete device, Delete user, Share device and Unshare device; (2) The methods/functions that do not have a common naming pattern, including Query device information, Update device data, and Access device.For the methods/functions in the former category, CAG can quickly identify them based on the common key

words/affixes used in them (as listed in Table III) via simple string matching. For the methods/functions in the latter category, we conduct static source code analyses to obtain the information of each method/function to determine whether its parameters or return values contains authorization-data. Notably, Natural language processing (NLP) can help to identify the method/function names, which is discussed in Section VII. Moreover, we build an
AST model for the source code to obtain the calling relationship of the methods/functions. With the calling relationship, we could remove (some of) the caller methods/functions from the
configuration file, since we only need to insert/weave the callee method/function for authorization-data protection.

E. Implementation of SecHARE

We present the implementation of SecHARE as follows with its source codes released online [16]. The configuration and CAG. As aforementioned, the configuration (provided by the user of SecHARE) specifies the names of the variables/methods/functions related to the
authorization-data. To help automatically generate the configuration file, CAG uses the QDox [46] to extract the definitions of the classes/interfaces/methods from the source code and uses Spoon [47] to build the AST model. Note that, the configuration also specifies the information needed to connect/access the database (e.g., the name of the database, the username and the password needed to connect the database), which is used to store the relationship between the authorization-data and shadow authorization-data. The CO: Taking the configuration file as input, CO generates the Aop.xml file in the format required by AspectJ [48]. The Aop.xml file would then be input to the DAA. Also, fromthe configuration file, CO extracts the names of relative variables and methods/functions and sends them to the PG. At
last, CO sends the database-related parameters (e.g., database username, password, etc.) to DO.
The DO: DO provides generalized database operation APIs, supporting both SQL and NoSQL databases. Currently, DO supports most SQL databases (a.k.a., Relational Database Man-
agement Systems) and the popular NoSQL database MongoDB [49]. The PG: Based on our defense (see Section V-B), we create the SECT to include all the possible behaviors needed
to insert/weave into the vulnerable IoT clouds. Specifically, we define code templates for data transferring, database read, database write and database deletion. Then, PG locates the vulnerable methods/functions in the original classes based on the input from CO, and automatically generates the patching codes using the templates in the SECT and the APIs pro-
vided by DO. Example-1 illustrates how PG patches the share Deivce() method. Specifically, shareDeivce() calls the getDevice() to obtain the authorization-data (e.g., device ID) and transmit the authorization-data to the delegatee user with thesendToDelegateeUser() method. PG patches such a progress in that: (1) adding line 10 to randomly generate the shadow authorization-data to ensure data uniqueness (in specific, we
used the RandomStringUtils.randomAlphanumeric() API [50] to generate the data); (2) adding line 11 to store the mapping relationship of the authorization-data, shadow authorization-data
and user’s identity; (3) replacing line 13 with line 12 to return the shadow authorization-data (instead of authorization-data). Note that, we maintain the data mapping at the user-level.
Since a single user typically possesses a limited number of devices, collisions between device mappings are expected to be infrequent. The DAA: The DAA is an AspectJ agent [51] that can be loaded into the running Java virtual machine. It takes inputs as the Aop.xml and the patching codes from PG to weave the patches into the original vulnerable classes when the Class Loader of the Java virtual machine loads the class files, forming the Security Enhanced Classes.

VI. EVALUATION

In this section, we discuss the impacts of authorization-data leakage flaws and evaluate the performance of SecHARE.

Example 1: Patching Sharedevice().

```
1  shareDevice (device, delegateeUser) {
2      ...
3      deviceID = getDevice(device, delegateeUser);
4      sendToDelegateeUser(deviceID, delegateeUser);
5      ...
6  }
7
8  getDevice(device, delegateeUser) {

9      ...
10     shadowData = generateShadowData(device.ID);
11     storeMapping(device.ID, shadowData,
   delegateeUser.ID);
12     return shadowData;
13     ~~return device.ID;~~

14 }
```

A. The Impacts of Authorization-Data Leakage

Prevalence of vulnerable authorization-data management: Bin et al. [5] identified several authorization-data leakage flaws in cross-cloud delegation, while we focused on the security issues of authorization-data management within a single IoT cloud. As shown in Table I, we identified 6 new flaws with 3 of them (Flaw 2, Flaw 4 and Flaw 5) affecting more than one IoT cloud, which shows the prevalence of the authorization-data leakage problem. Scope of the impact: The 4 open-source IoT clouds we analyzed (i.e., ThingsBoard [13], JetLinks [15], Kaa open-source [14] and ThingsPanel [52]) are among the most popular IoT projects in the open-source community, with over 17 K stars on GitHub in total. The other 2 commercial IoT cloud platforms (i.e., Kaa Enterprise [36] and ThingsKit [38]) serve many enterprises (including Lenovo, Alibaba cloud and NET4.IO [36], [38]) and customers, and connect millions of devices in various field (e.g., smart energy, smart agriculture, smart home, and industrial Internet of Things [53], [54]). Therefore, security loopholes in these IoT clouds can bring huge damage to the real world IoT applications.

B. Performance Evaluation

Selecting IoT clouds for flaw identification: Since we focused on the security issues in the IoT device sharing within a single cloud, we only studied the clouds that support such functionality and enforce access control mechanisms. Also, we prioritized the general IoT clouds — the clouds can be applied to multiple IoT scenarios (e.g., smart home, smart city, smart energy, etc.). At last, we prioritized the clouds with better popularity — more GitHub stars for the open-source clouds and more customers for the commercial clouds. Selecting IoT platforms for defense evaluation: SecHARE fixes the flaws by patching the source codes of the clouds. Hence,we only evaluated SecHARE upon the open-source clouds. Further, multiple programming languages (e.g., Java, Go, C++, C, etc.) are used to implement the open-source IoT clouds.

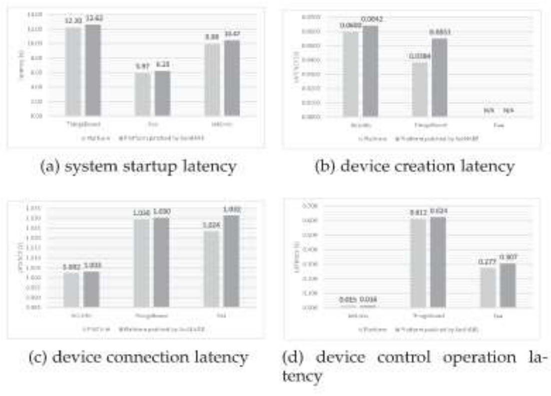

(a) system startup latency (b) device creation latency
(c) device connection latency (d) device control operation latency

Fig. 12. Cost assessment of *SecHARE* applied to 3 cloud platforms (Kaa open-source platform cannot accurately measure the latency of creating a device).

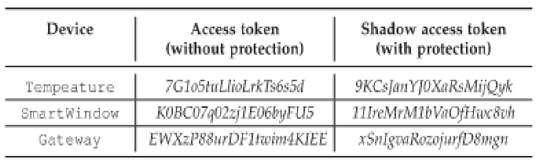
However, according to the Eclipse Foundation IoT survey [55], Java is the top choice with a popularity of 66.5%. Therefore, we applied SecHARE to the three open-source IoT clouds written in Java (e.g, ThingsBoard, JetLinks, and Kaa open-source). Efficiency: To evaluate the efficiency of SecHARE, we deployed 3 popular open-source IoT platforms (i.e., ThingsBoard, Kaa open-source, and JetLinks) in our test server (with Intel Core i7-9700 cpu, 16 GB memory). With each cloud, we carried out multiple operations (including system startup, device creation, device connection, and device control) before and after it is patched by SecHARE (experimental programs and data are publicly available at [16]). We repeated the system startup operation for 20 times and measured the time. As shown in Fig. 12(a), the overhead on startup time introduced by SecHARE is 400 ms averagely. For the device creation, device connection and device control operations, we repeated the experiments for 2000 times. As shown in Fig. 12(b), (c) and (d), the average overheads are 10.39 ms, 3.17 ms and 14.25 ms respectively. We believe such overheads is negligible. Performance overheads: In order to assess the impact of deploying SecHARE on a real-world cloud platform, we also conducted a series of performance evaluations on our test server. Specifically, we measured the CPU and run-time memory usage for 1000 device creation and data querying operations on the ThingsBoard both before and after deploying SecHARE, respectively. We observed an increase of only 0.14% in CPU usage and 0.16% in memory usage, indicating that the performance overheads introduced by SecHARE is negligible.

C. Security Benefit

As discussed in Section IV, the attacker can leverage the leaked authorization-data to communicate with the cloud even after his access right is revoked. We evaluated whether the attacker can achieve that in the cloud that has been patched by SecHARE. Specifically, we set up 3 different devices: the temperature sensor, the smart window, and the gateway device. Each device was assigned a specific operation, such as uploading device data/events, receiving remote control commands, and managing gateway sub-devices. As depicted in Table IV, we ensured that the authorized user did not have access to the actual authorization-data, which remained undisclosed to them within the SecHARE-patched cloud. Next, we assessed the scenario in which a malicious user (e.g., an attacker), possessing retained authorization-data (access token), attempts to exploit vulnerabilities in an unpatched cloud platform, as illustrated in Table V. Through our evaluation, we

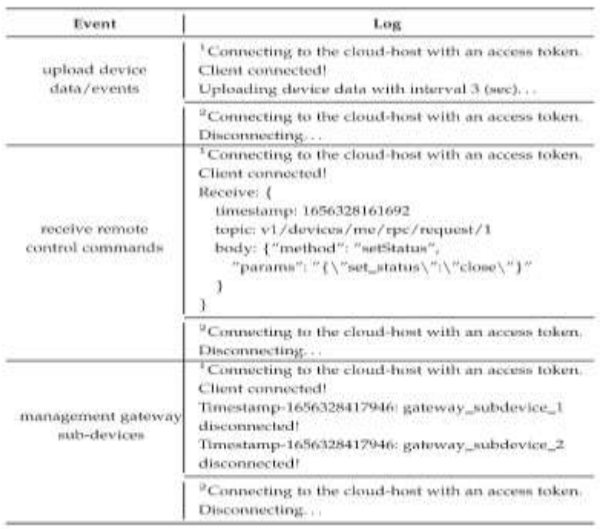
observed that the attacker could engage in data forgery attacks by uploading device data, privacy leakage attacks by receiving remote control commands, and denial-of-service attacks by disconnecting/logging out the gateway device, thereby disrupting the service of sub-devices.
However, when these operations were attempted within the SecHARE-patched cloud platform, the system effectively denied all unauthorized access attempts, preventing harm caused by the leakage of authorization-data. This indicates that our proposed defense can effectively mitigate the flaws.

TABLE IV
AUTHORIZATION-DATA TRANSMITTED TO THE USER WITH AND WITHOUT OUR PROTECTION



Device	Access token (without protection)	Shadow access token (with protection)
`Tempeature`	*7G1o5tuLlioLrkTs6s5d*	*9KCsJanYJ0XaRsMijQyk*
`SmartWindow`	*K0BC07q02zj1E06byFU5*	*11IreMrM1bVaOfHwc8vh*
`Gateway`	*EWXzP88urDF1twim4KIEE*	*xSnIgvaRozojurfD8mgn*

TABLE V
LOGS OUTPUT BY THE CLOUD PLATFORM UNDER DIFFERENT OPERATIONS BEFORE AND AFTER THE PATCH



Event	Log
upload device data/events	[1]Connecting to the cloud-host with an access token. Client connected! Uploading device data with interval 3 (sec)...
	[2]Connecting to the cloud-host with an access token. Disconnecting...

receive remote control commands	[1]Connecting to the cloud-host with an access token.
	Client connected!
	Receive: {
	  timestamp: 1656328161692
	  topic: v1/devices/me/rpc/request/1
	  body: {"method": "setStatus",
	    "params": "{\"set_status\":\"close\"}"
	  }
	}

	[2]Connecting to the cloud-host with an access token. Disconnecting...
management gateway sub-devices	[1]Connecting to the cloud-host with an access token. Client connected! Timestamp-1656328417946: gateway_subdevice_1 disconnected! Timestamp-1656328417946: gateway_subdevice_2 disconnected!
	[2]Connecting to the cloud-host with an access token. Disconnecting...

VII. DISCUSSION AND FUTURE WORK

Manual efforts to secure an IoT cloud: As discussed in Section V-C, SecHARE requires the user to provide a configuration file. Specifying the configuration file requires manual efforts. Although we developed CAG to reduce such manual efforts, certain efforts are still needed when CAG is not able to determine the exact methods/functions. Towards fully automated analyses: To further improve the automation of SecHARE, NLP techniques can be used to automatically locate/identify the method/function names in the source code. By parsing functions and extracting features from the source code, NLP can make SecHARE more accurate and efficient. Therefore, in future work, we aim to explore the feasibility and effectiveness of integrating NLP techniques into SecHARE to improve its automation and accuracy. Protection of cross-cloud device sharing: Although we only applied SecHARE to secure the device sharing within a single IoT cloud, our general defense can also help to secure the cross- cloud device sharing. For example, Bin et al. [5] found that the deviceID of the SmartThings device (which is treated as a credential in SmartThings) could be leaked to a malicious delegatee user in the Google Home. Leveraging the leaked deviceID, the malicious user can control the victim's SmartThings devices that he is not entitled to access. This problem can be also fixed with our data mapping scheme. When the SmartThings transmits the
deviceID to the Google Home, the SmartThings could generate a new deviceID (denoted as deviceID') and send the deviceID' to Google Home. Upon receiving a request from Google Home carrying deviceID', the SmartThings can transfer the deviceID' to deviceID, and perform authorization check based on deviceID. When revoking the access right of Google Home, SmartThings can delete the deviceID', thus to fix the problem without disrupting the normal IoT service. Note that, SmartThings should NOT
delete deviceID, since it is also used by other users/applications in the SmartThings. Also, SmartThings can NOT refuse to send the identifier of the delegated device to the Google Home, since the access delegation protocol of Google Home requires such information. Supporting more languages: Diverse programming languages, including Java, Go, and C#, are employed in the implementation of contemporary IoT clouds. Presently, SecHARE
has adopted the AspectJ framework specifically to support Java programming language. Notably, analogous frameworks are available for other programming languages, such as GoAOP or Go-Aspect for Go, and AspectDNG for C#. In future work, we aim to explore the applicability of these frameworks to accommodate diverse programming languages. It is worth noting that the fundamental concept underlying our proposed defense mechanism is general in nature, thus facilitating its extension to other IoT clouds.

VIII. RELATED WORK

IoT platform security: In the rapid development of the IoT, the IoT cloud plays an important role. Chen et al. [56] and Zhou et al. [57] have reported flaws found in device management for IoT clouds, demonstrating that leakage of device
identity can have serious consequences. However, they only discovered the vulnerabilities without proposing any defense
mechanisms. Yuan et al. [5] proposed a semi-automated

tool to detect cross-cloud IoT delegation vulnerabilities. In contrast, our work focuses on authorization issues within individual cloud platforms and provides an automated protection tool (SecHARE) to mitigate the authorization-data leakage problem. Moreover,
most of the existing work is mainly for specific platforms, such as SmartThings [7], [9], [58], [59], [60], [61], [62], [63], [64], [65], IFTTT [10], [66], [67] and AWS Alexa [68], [69]. By contrast, our work is to provide a tool to protect different cloud platforms. Besides that, some works [7], [62], [66], [70] provide methods to
protect sensitive information or data flow in IoT apps, whereas our work is focuses on protecting authorization-data only in the cloud. IoT permission sharing: Permission issues have always been one of the key concerns of IoT security and have been widely studied [9], [10], [11], [58], [59], [71], [72], [73], [74]. Fernandes
et al. [9] first reported that the coarse-grained capability design leads to over-privileged and the inability of the event subsystem to adequately protect events carrying sensitive information in Smart Things. Additionally, access control is not only distributed but also heterogeneous and ad-hoc in today’s IoT cloud
ecosystem. To cope with the new application scenario, Jia et al. [58] focused on permission protection and proposed ContexIoT, a fine-grained context-based permission system for SmartThings to provide context integrity for IoT programs at runtime. Tian et al. [59] presented a user-centric, semantic-based authorization design called SmartAuth to help users avoid overly privileged applications in SmartThings. These researches primarily focus on the permission management of the applications, without consideration of dynamic user authorization scenarios or proposing methods to secure the authorization-data. Fernandes et al. [62] proposed a privacy-preserving system called FlowFence, which attempts to address the ineffectiveness of existing permission-based access controls in controlling sensitive data flows in applications by embedding the data flow patterns expected by users. However, this work mainly tries to prevent malicious IoT applications from abusing the sensitive data (e.g., data collected by the IoT sensors). In con-
trast, SecHARE focuses on securing the data used for authorization and preventing unauthorization access in a shared IoT scenario.
Furthermore, Fernandes et al. [10] introduced Decentralized Action Integrity to prevent an untrusted trigger-action platform from misusing compromised OAuth tokens. Andersen et al. [11] presented WAVE, an authorization frame-
work offering decentralized trust, which supports transitive fine-grained sharing and revocation. However, these efforts, while meeting the current complex IoT authorization needs, require all parties to work together following the same framework APIs and are more difficult to apply and deploy to the real world. In contrast, our work only adds a few changes to the cloud platform to realize automatic protection of authorization-data. Moreover, our tool can adapt to a variety of authorization-data and is compatible with different cloud platforms.

IX. CONCLUSION

In this paper, we systematically study how the authorization data are managed in the real-world IoT device sharing and its security implications. Our research reveals that authorizationdata leakage is prevalent in the IoT clouds, with 6 flaws identified in 6 popular IoT clouds. To mitigate the problem, we proposed SecHARE to automatically patch the vulnerable codes of the
IoT clouds. We applied SecHARE to 3 open-source IoT clouds. Our evaluation shows that SecHARE is easy to use by the IoT vendors, effective and efficient in securing authorization-data. Our new understanding and new techniques will provide better protection for today’s IoT cloud platforms, as well as those to
be built in the years to come.

REFERENCES

[1] “AWS IoT,” 2023. Accessed: Mar. 2023. [Online]. Available: https://aws.amazon.com/iot/
[2] “SmartThings Samsung,” 2023. Accessed: Mar. 2023. [Online]. Available: https://www.smartthings.com/
[3] “Philips HUE,” 2023. Accessed: Mar. 2023. [Online]. Available: https://www2.meethue.com/
[4] “MiHome,” 2023. Accessed: Mar. 2023. [Online]. Available: https:// xiaomi-mi.com/mi-smart-home/
[5] B. Yuan et al., “Shattered chain of trust: Understanding security risks in cross-cloud IoT access delegation,” in Proc. 29th USENIX Secur. Symp.,2020, pp. 1183–1200.
[6] “OAuth 2.0,” 2023. Accessed: Mar. 2023. [Online]. Available: https://oauth.net/2/
[7] Z. B. Celik et al., “Sensitive information tracking in commodity IoT,” inProc. 27th USENIX Secur. Symp., 2018, pp. 1687–1704.
[8] Y. Sameshima and P. T. Kirstein, “Authorization with security attributes and privilege delegation: Access control beyond the ACL,” Comput. Commun., vol. 20, no. 5, pp. 376–384, 1997.
[9] E. Fernandes, J. Jung, and A. Prakash, “Security analysis of emerging smart home applications,” in Proc. IEEE 37th Symp. Secur. Privacy, 2016,pp. 636–654.

[10] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decentralized action integrity for trigger-action IoT platforms," in Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp., 2018.
[11] M. P. Andersen et al., "WAVE: A decentralized authorization framework with transitive delegation," in Proc. 28th USENIX Secur. Symp., 2019, pp. 1375–1392.
[12] "Chinese national vulnerability database," Accessed: Mar. 2023.[Online]. Available: https://en.wikipedia.org/wiki/Chinese_National_ Vulnerability_Database
[13] "Thingsboard Github," 2023. Accessed: Mar. 2023. [Online]. Available: https://github.com/thingsboard/thingsboard
[14] "Kaa Github," 2023. Accessed: Mar. 2023. [Online]. Available: https:// github.com/kaaproject/kaa
[15] "JetLinks Github," 2023. Accessed: Mar. 2023. [Online]. Available: https://github.com/jetlinks/jetlinks-community
[16] "SecHARE," 2023. Accessed: Mar. 2023. [Online]. Available: https://github.com/SecHARE/SecHARE
[17] "Zigbee | complete IOT solution," 2023. Accessed: Mar. 2023. [Online].Available: https://csa-iot.org/all-solutions/zigbee/
[18] "Better and safer smart homes are built on Z-wave," 2023. Accessed:Mar. 2023. [Online]. Available: https://www.z-wave.com/
[19] "Bluetooth low energy (BLE)," 2023. Accessed: Mar. 2023. [On-line]. Available: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/
[20] "MQTT | The standard for IoT messaging," 2023. Accessed: Mar. 2023.[Online]. Available: https://mqtt.org/
[21] "HTTP | Hypertext transfer protocol," 2023. Accessed: Mar. 2023. [On-ine]. Available: https://www.w3.org/Protocols/
[22] "CoAP | Constrained application protocol," 2023. Accessed: Mar. 2023.[Online]. Available: http://coap.technology/
[23] "M2M lightweight (LWM2M)," 2023. Accessed: Mar. 2023. [On-line]. Available: https://omaspecworks.org/what-is-oma-pecworks/iot/lightweight-m2m-lwm2m/
[24] "Advanced message queuing protocol (AMQP)," 2023. Accessed:
Mar. 2023. [Online]. Available: https://www.amqp.org/
[25] Y. Jia et al., "Burglars' IoT paradise: Understanding and mitigating security risks of general messaging protocols on IoT clouds," in Proc. IEEE 41st Symp. Secur. Privacy, 2020, pp. 465–481.
[26] "Publish–subscribe pattern," 2023. Accessed: Mar. 2023. [Online]. Available: https://en.wikipedia.org/wiki/Publish-subscribe_pattern
[27] "HomeKit," 2023. Accessed: Mar. 2023. [Online]. Available: https://developer.apple.com/documentation/homekit
[28] "Google home," 2023. Accessed: Mar. 2023. [Online]. Available: https://developers.google.com/assistant/smarthome/overview
[29] "IFTTT," 2023. Accessed: Mar. 2023. [Online]. Available: https://ifttt.com/
[30] G. Kiczales et al., "Aspect-oriented programming," in Proc. 11th Eur.Conf. Object-Oriented Program., 1997, pp. 220–242.
[31] "AspectJ," 2023. Accessed: Mar. 2023. [Online]. Available: https://www.eclipse.org/aspectj/doc/released/
[32] "Spring AOP," 2023. Accessed: Mar. 2023. [Online]. Available: https://docs.spring.io/spring-framework/docs/2.5.5/reference/aop.html
[33] "Alibaba cloud IoT," 2023. Accessed: Mar. 2023. [Online]. Available:https://www.alibabacloud.com/product/iot
[34] "Tuya," 2023. Accessed: Mar. 2023. [Online]. Available: https://en.tuya. com/solutions
[35] "ThingsBoard," 2023. Accessed: Mar. 2023. [Online]. Available: https: //thingsboard.io/
[36] "Kaa," 2023. Accessed: Mar. 2023. [Online]. Available: https://www.kaaiot.com/
[37] "JetLinks," 2023. Accessed: Mar. 2023. [Online]. Available: https://www.jetlinks.cn/
[38] "ThingsKit," 2023. Accessed: Mar. 2023. [Online]. Available: https://www.thingskit.com/
[39] "ThingsPanel," 2023. Accessed: Mar. 2023. [Online]. Available: https://www.thingspanel.cn/
[40] W. He et al., "Rethinking access control and authentication for the home Internet of Things (IoT)," in Proc. 27th USENIX Secur. Symp., USENIX Secur., 2018, pp. 255–272.
[41] Y. Jia et al., "Who's in control? on security risks of disjointed IoT device management channels," in Proc. 28th ACM SIGSAC Conf. Comput.Commun. Secur., 2021, pp. 1289–1305.
[42] "MQTTX," 2023. Accessed: Mar. 2023. [Online]. Available: https:// github.com/emqx/MQTTX
[43] "Postman," 2023. Accessed: Mar. 2023. [Online]. Available: https://www. postman.com/
[44] "Wireshark," 2023. Accessed: Mar. 2023. [Online]. Available: https: //www.wireshark.org/
[45] "JetLinks official protocol document," 2023. Accessed: Mar. 2023.[Online]. Available: http://doc.jetlinks.cn/basics-guide/jetlinks-protocol-support.html
[46] "QDox," 2023. Accessed: Mar. 2023. [Online]. Available: https://github.com/paul-hammant/qdox
[47] "Spoon," 2023. Accessed: Mar. 2023. [Online].

Available: https://github.com/INRIA/spoon
[48] "Configuration of load-time weaving in AspectJ," 2023. Accessed:Mar. 2023. [Online]. Available: https://www.eclipse.org/aspectj/doc/ released/devguide/ltw-configuration.html
[49] "MongoDB," 2023. Accessed: Mar. 2023. [Online]. Available: https://www.mongodb.com/home
[50] "RandomStringUtils.randomAlphanumeric()," 2023. Accessed:Mar. 2023. [Online]. Available: https://commons.apache.org/proper/commons-lang/javadocs/api-3.1/org/apache/commons/lang3/ RandomStringUtils.html#randomAlphanumeric(int)
[51] "LTW weavingagents," 2023. Accessed: Mar. 2023. [Online]. Available:https://www.eclipse.org/aspectj/doc/released/ devguide/ltw-agents.html
[52] "ThingsPanel github," 2023. Accessed: Mar. 2023. [Online]. Available:https://github.com/ThingsPanel/ThingsPanel-Go
[53] "Kaa IoT use cases," 2023. Accessed: Mar. 2023. [Online]. Available:https://www.kaaiot.com/use-cases
[54] "ThingsKit industry solutions," 2023. Accessed: Mar. 2023. [Online].Available: https://www.thingskit.com/portfolio
[55] "IoT developer survey2018," 2023. Accessed: Mar. 2023. [Online]. Available: https://www.slideshare.net/kartben/iotdeveloper-survey-2018
[56] J. Chen et al., "Your IoTs are (Not) mine: On the remote binding between IoT devices and users," in Proc. IEEE/IFIP 49th Annu. Int. Conf. Dependable Syst. Netw., 2019, pp. 222–233.
[57] W. Zhou et al., "Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms," in Proc. 28th USENIX Secur. Symp., 2019, pp. 1133–1150.
[58] Y. J. Jia et al., "ContexloT: Towards providing contextual integrity to appified IoT platforms," in Proc. 24th Annu. Netw. Distrib. Syst. Secur. Symp., 2017, pp. 1–15.
[59] Y. Tian et al., "SmartAuth: User-centered authorization for the Internet of Things," in Proc. 26th USENIX Secur. Symp., 2017, pp. 361–378.
[60] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT," in Proc. 26th Annu. Netw.Distrib. Syst. Secur. Symp., 2019, pp. 1–15.
[61] W. Ding and H. Hu, "On the safety of IoT device physical interaction control," in Proc. 25th ACM SIGSAC Conf. Comput. Commun. Secur.,2018, pp. 832–846.
[62] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "FlowFence: Practical data protection for emerging IoT application frameworks," in Proc. 25th USENIX Secur. Symp., 2016,pp. 531–548.
[63] Q. Wang, W. U. Hassan, A. Bates, and C. A. Gunter, "Fear and logging in the Internet of Things," in Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp., 2018, pp. 1–15.
[64] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "HoMonit:Monitoring smart home apps from encrypted traffic," in Proc. 25th ACM SIGSAC Conf. Comput. Commun. Secur., 2018, pp. 1074–1088.
[65] B. Yuan et al., "SmartPatch: Verifying the authenticity of the trigger-event in the IoT platform," IEEE Trans. Dependable Secure. Comput., vol. 20, no. 2, pp. 1656–1674, Mar./Apr. 2023.
[66] I. Bastys, M. Balliu, and A. Sabelfeld, "If this then what?: Controlling flows in IoT apps," in Proc. 25th ACM SIGSAC Conf. Comput. Commun.Secur., 2018, pp. 1102–1119.
[67] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, and C. A. Gunter, "Charting the attack surface of trigger-action IoT platforms," in Proc. 26th ACM SIGSAC Conf. Comput. Commun. Secur., 2019, pp. 1439–1453.
[68] N. Zhang, X. Mi, X. Feng, X. Wang, Y. Tian, and F. Qian, "Dangerous skills: Understanding and mitigating security risks of voice-controlled third-party functions on virtual personal assistant systems," in Proc. IEEE 40th Symp. Secur. Privacy, 2019, pp. 1381–1396.
[69] L. Cheng, C. Wilson, S. Liao, J. Young, D. Dong, and H. Hu, "Dangerous skills got certified: Measuring the trustworthiness of skill certification in voice personal assistant platforms," in Proc. 27th ACM SIGSAC Conf.
Comput. Commun. Secur., 2020, pp. 1699–1716.[70] X. Li, J. Li, S. Yiu, C. Gao, and J. Xiong, "Privacy-preserving edge-assisted
image retrieval and classification in IoT," Front. Comput. Sci., vol. 13,no. 5, pp. 1136–1147, 2019.
[71] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Security implications of permission models in smart-home application frameworks," IEEE Secur. Privacy, vol. 15, no. 2, pp. 24–30, Mar./Apr. 2017.
[72] A. Rahmati, E. Fernandes, K. Eykholt, and A. Prakash, "Tyche: A risk-based permission model for smart homes," in Proc. IEEE 3rd Cybersecu-rity Develop., 2018, pp. 29–36.
[73] W. Wu, S. Hu, D. Lin, and G. Wu, "Reliable resource allocation with RF fingerprinting authentication in secure IoT networks," Sci. China Inf. Sci.,vol. 65, no. 7, pp. 1–16, 2022.
[74] Z. Guan, W. Yang, L. Zhu, L. Wu, and R. Wang, "Achieving adaptively secure data access control with privacy protection for lightweight IoTdevices," Sci. China Inf. Sci., vol. 64, no. 6, 2021, Art. no. 162301