Hidden and Lost Control: on Security Design Risks in IoT User-Facing Matter Controller

Haoqiang Wang*†‡¶, Yiwei Fang*†‡¶, Yichen Liu†, Ze Jin*†‡, Emma Delph†, Xiaojiang Du§, Qixu Liu*‡‖, Luyi Xing†‖
*Institute of Information Engineering, Chinese Academy of Sciences
†Indiana University Bloomington
‡School of Cyber Security, University of Chinese Academy of Sciences
§Stevens Institute of Technology
{wanghaoqiang, fangyiwei, jinze, liuqixu}@iie.ac.cn, {xdu16}@stevens.edu, {liuyic, emdelph, luyixing}@iu.edu

***Abstract*—Matter is emerging as an IoT industry–unifying standard, aiming to enhance the interoperability among diverse smart home products, enabling them to work securely and seamlessly together. With many popular IoT vendors increasingly supporting Matter in consumer IoT products, we perform a systematic study to investigate how and whether vendors can integrate Matter securely into IoT systems and how well Matter as a standard supports vendors' secure integration.**

By analyzing Matter development model in the wild, we reveal a new kind of design flaw in user-facing Matter control capabilities and interfaces, called *UMCCI* flaws, which are exploitable vulnerabilities in the design space and seriously jeopardize necessary control and surveillance capabilities of Matter-enabled devices for IoT users. Therefore we built an automatic tool called *UMCCI* Checker, enhanced by the large-language model in UI analysis, which enables automatically detecting *UMCCI* flaws without relying on real IoT devices. Our tool assisted us with studying and performing proof-of-concept attacks on 11 real Matter devices of 8 popular vendors to confirm that the *UMCCI* flaws are practical and common. We reported *UMCCI* flaws to related vendors, which have been acknowledged by CSA, Apple, Tuya, Aqara, etc. To help CSA and vendors better understand and avoid security flaws in developing and integrating IoT standards like Matter, we identify two categories of root causes and propose immediate fix recommendations.

I. Introduction

The development of Matter, initially known as Project CHIP (Connected Home over IP), was launched by the Connectivity Standards Alliance (CSA), formerly known as the Zigbee Alliance. The primary objective of the Matter project is to develop a universal, open-source application-layer connectivity standard that can overcome the prevalent fragmentation and heterogeneity in the smart home industry. Emerging as an industry–unifying standard, Matter aims to enhance the interoperability among diverse smart home products, enabling them to work securely and seamlessly together. Since late 2022 [4], major IoT vendors (Apple, Google, Amazon, SmartThings, Tuya, etc.) have extensively deployed support for Matter in their IoT devices, IoT mobile apps (e.g., Google Home, SmartThings, Apple Home, Amazon Alexa), and IoT development frameworks (e.g., HomeKit [10], Tuya SDK [14], etc.). With this trend, the number of users using the Matter technology is growing rapidly worldwide.

¶The first two authors Haoqiang Wang and Yiwei Fang contributed equally to this work and they were intern students at Indiana University Bloomington at the time the work was conducted.
‖Corresponding author: Luyi Xing, Qixu Liu.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.240266
www.ndss-symposium.org

Matter open-source development and vendor integration. At a high-level, Matter is known to be an open-source standard with public specifications. While the Matter standard internally includes protocol specification for multiple IoT functionalities, including device commissioning, user management, local access control, etc., we refer to them collectively as the *Matter protocol*. The Matter standard and protocol comes with an open-source implementation, as a project named "connectedhomeip" [31] maintained on GitHub by its open-source community coordinated by CSA. The open-source developers include those from industry members of CSA [18] such as Apple, Google, and Comcast and individual developers of interest. They contribute to the open-source layer of a Matter-enabled product. Moreover, individual vendors develop specific applications and features on top of the open-source layer of Matter (see the Matter development model, § II-B). With popular IoT vendors increasingly supporting Matter in consumer IoT products, we ask three key research questions to guide our research. (**RQ1**:) *How do real-world IoT vendors integrate Matter to existing IoT devices and applications?* (**RQ2**:) *Can vendors securely integrate Matter to IoT devices and applications, and what is the error space that we should be aware of?* (**RQ3**:) *How well does Matter as a standard define and support IoT vendors' secure integration?*

Revealing security risks in vendors' Matter integration. By analyzing the Matter development model in the wild which includes three layers: the Matter open-source layer, and the vendor's local and cloud layers (Figure 2 in § II-B), we uncover the security responsibilities of multiple parties in developing Matter and integrating Matter into real IoT devices, including the open-source community and the vendors (§ III). Specifically, the open-source community [31] provides implementation for (1) device firmware and (2) the

Matter protocol to control devices. In particular, the latter is implemented as an SDK called the CHIP SDK [30], which implements a range of primitive capabilities to communicate with and control the Matter-enabled IoT devices (called *Matter devices*). On top of the Matter open-source implementation, vendors develop *Matter controllers* (usually proprietary), by integrating the CHIP SDK and using the SDK APIs to access and control *Matter devices*. Matter controller is a logical concept in Matter, being transparent to users, and they are commonly implemented either as part of a mobile app (e.g., Tuya Smart [15] and uHome+ [16] app) or inside an IoT gateway hub especially in smart speakers (Figure 2), e.g., Apple HomePod [10], Google Nest Mini [7], Amazon Echo Dot [6]. In the IoT mobile app or hub, vendors additionally develop user-facing Matter control capabilities and interfaces (*UMCCI*), as graphical user interface (GUI) or voice interface, for users to view, use, and control Matter devices through Matter controllers (§ II-A).

We find that the design and practice of *UMCCI* are up to individual IoT applications, devices, and Matter functionalities that the vendors support. Although the Matter standard focused a lot on standard functionalities, low-level data model, etc., particularly helpful for the open-source layer development of Matter (the firmware and CHIP SDK, Figure 2), the standard lacks guidance and security design to help vendors securely integrate Matter. *In particular, how to design and develop UMCCI providing secure and sufficient Matter control capabilities for users, is up to individual vendors, which can easily go wrong and comes with a range of serious design flaws violating three security requirements (P1-P3) we summarized for vendors to integrate an application-layer IoT standard like Matter* (§ III).

For example, we expect that the vendors' IoT app should show users all Matter controllers that have control (i.e., access rights) over their IoT devices — unawareness of unauthorized control poses serious security threats to legitimate users [48].[1] However, we find that the apps of Apple Home, SmartThings, Google Home, Tuya, and others either show a partial list of controllers based on the vendors' problematic interpretation of Matter *UMCCI*, or show unverified controller information (e.g., controller vendors or controller names) that can be faked by malicious IoT users (such as prior employees, guests, and delegatee users [39], [40], [43], [58], [62]) to hide from device-owners their unauthorized control over Matter devices. Those design flaws in the wild, termed *UMCCI* flaws in our study, seriously endanger Matter users' control and awareness capabilities for Matter devices. By studying Matter integration of 11 popular IoT devices from 8 vendors, we report six types of novel *UMCCI* flaws (Table II), which enable practical exploits that allow unauthorized users to control Matter devices through covert channels or hidden Matter controllers, or even allow unauthorized users to arbitrarily downgrade or remove device owners' privileges. We performed proof-of-concept attacks on 11 real Matter devices of 8 vendors to confirm that the *UMCCI* flaws are real and practical (see attack demo online [13]).

Automatic detection of *UMCCI* flaws. Our Matter security analysis is assisted by a novel automatic tool called *UMCCI* Checker (§ VI), that is based on large-language model (LLM) assisted automatic UI analysis and is capable of detecting *UMCCI* flaws from the IoT vendors' apps. *UMCCI* Checker automatically detects *UMCCI* flaws without relying on purchasing and configuring real IoT devices, which is made possible by a virtual Matter device we developed armed with *UMCCI* attack vectors that can automatically pair and cooperate universally with different vendors' Matter controllers (§ VI-A). The automatic pairing of our virtual Matter device with vendor apps featuring diverse GUI workflows is enabled by our LLM-enhanced UI automation technique as part of *UMCCI* Checker. Our evaluation shows that *UMCCI* Checker is highly effective, efficient and scalable. We reported the *UMCCI* flaws in Table II to related vendors and all *UMCCI* flaws we reported were acknowledged by CSA, which maintains Matter standards and is seeking modifications to Matter specifications to regulate vendors' *UMCCI* implementations and help mitigate the problems. Apple and Aqara told us they would fix these flaws promptly by providing view and control capabilities meeting our security requirements $P1$-$P3$. Tuya has already fixed the problem based on our suggestion.

Towards defense against *UMCCI* flaws. To help CSA and vendors better understand and avoid security flaws in developing and integrating IoT standards like Matter, we systematize the *UMCCI* flaws in two categories based on a responsibility model that determines whether the flaw is caused by vendors (§ IV) or due to flaws in the Matter standard (§ V). In the two respective categories, we show that *UMCCI* flaws come with two root causes: (1) vendors' ad-hoc, insecure practices in developing user-facing Matter control capabilities, (2) inadequate security design in the underlying Matter protocol for ensuring user-facing Matter control information (e.g., controllers, fabrics, ACL) is trustworthy and the control is adequate (e.g., revoking control). Our root cause analysis shows that Matter aims to provide uniform, cross-vendor interoperable control for end-users by defining rich application-layer semantics and data models (e.g., fabrics, controllers, Vendor IDs/Labels), unlike protocols like ZigBee/Bluetooth [56], which focus more on networking. *UMCCI* flaws show that complex application-level semantics give vendors room for logic errors, introducing new attack vectors that are hard to prevent. Moreover, *UMCCI* flaws reveal complex security and development responsibilities between vendors and the Matter standard. *Our findings call for immediate and thorough security analysis of the Matter protocol, the Matter open-source implementation (CHIP SDK), and vendor implementation (atop CHIP SDK).* We discuss immediate changes that should be made to the Matter standards and vendor practices to fix *UMCCI* flaws in § VI and Appendix B.

Contributions. We summarize our contributions as follows:

• *New understanding and new attacks*. We perform a novel, systematic security analysis for Matter's development model to understand (1) how vendors integrate Matter standard to IoT devices and applications; (2) the error space of vendors' Matter integration; (3) novel root causes of security risks in vendors' integration of Matter. Our study reveals serious design flaws unknown before, termed *UMCCI* flaws, that seriously jeopardize necessary control and surveillance capabilities of Matter devices for IoT users. We discovered *UMCCI* flaws from 8 top-of-the-line IoT vendors that integrated the Matter standard, performed PoC attacks with 11 real devices, showing that *UMCCI* attacks are realistic and *UMCCI* flaws are

[1]The term "app" in this paper always refers to the IoT vendor's mobile app.

common. All the *UMCCI* flaws are acknowledged by CSA.

• *New detection and mitigation techniques*. We have developed novel techniques for pinpointing *UMCCI* flaws from real vendors without relying on purchasing and physically configuring real devices, mitigating a prior scalability problem thwarting practical security analysis across IoT vendors. Based on our findings, we provide recommendations for vendors and CSA respectively to fix *UMCCI* flaws. More importantly, we identify two categories of root causes to help CSA and vendors better avoid *UMCCI* flaws and related security risks in future development and integration of IoT standard.

II. Background

A. Basic Concepts in the Matter Protocol

While the Matter standard internally includes protocol specification for multiple IoT functionalities, including device commissioning, user management, local access control, etc., we refer to them collectively as the *Matter protocol*. In this study, we focus on the Matter protocol at the application layer, without emphasis on the networking layers that Matter is built on, including the Internet Protocol (IP), Thread and Wi-Fi network transports. In the following, we elaborate on key, basic concepts in the Matter protocol. The detailed specification can also be found at the *Matter Handbook* [5] and *Matter Specification* version 1.3 [34].

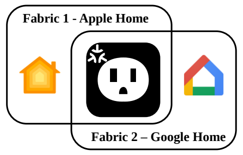


Fig. 1: A Matter Plug in Two Fabrics

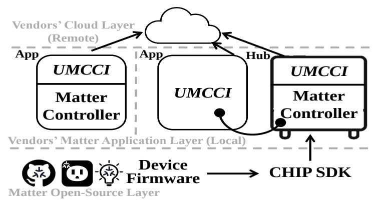


Fig. 2: Matter Development Model

• *Cluster:* A cluster represents a specific set of functionalities or attributes maintained inside Matter devices. For example, the cluster named "On/Off" can be used in a light bulb, and setting the cluster value as either "on" or "off" effectively toggles switch of the bulb.

• *Node:* A Matter node represents an addressable resource, each possessing a unique address within the same Matter network. For instance, a Matter-supported plug or an app on a mobile phone acting as a Matter controller, both constitute Matter nodes, each with its unique Node ID.

• *Fabric:* Fabric refers to a collection of Matter devices and controllers, which form a virtual network of Matter nodes. A node can simultaneously join multiple fabrics. Each fabric contains multiple Matter attributes such as a Vendor ID, Label, and others, designated by a Matter controller (see below) that creates the fabric by pairing with the device through the Matter pairing code (Figure 1). Devices within a fabric share the same Certificate Authority (CA) top-level certificate and a unique 64-bit identifier named Fabric ID. Within a fabric, each device node possesses a unique NOC certificate (see below).

• *Controller:* A Matter controller is an entity (i.e., a Matter node) that can control Matter devices paired with it. Matter controller functionality can be integrated into hardware devices such as IoT hubs or mobile apps. Multiple Matter controllers can exist within a fabric to offer redundancy and convenient control options for users.

• *Matter Certificates:* The Matter protocol uses three types of certificates to authenticate and authorize nodes, conceptually administrators, low-privilege users, and devices. Each fabric has a self-signed Root Certificate Authority Certificate (RCAC). Within the fabric, Matter devices are identified by a Node Operational Credential (NOC) or an optional Intermediate Certificate Authority Certificate (ICAC). The ICAC, signed by the RCAC, can issue the NOC. If there is no ICAC, the NOC is signed directly by the RCAC.

• *Fabric Administrator:* Fabric administrators are usually Matter controllers that possess the ICAC or RCAC of a fabric, enabling them to add devices to the fabric as new Matter nodes by issuing NOCs, as well as remove devices from the fabric.

• *On-Device in-Fabric Access Control List (ACL):* Each Matter device has an ACL cluster that maintains an ACL for each fabric of the device, recording the privileges of other Matter nodes in the fabric to perform operations on the device. Based on the ACL, operation requests from unauthorized nodes are denied by the device. Privileges, ranked from low to high, include View, ProxyView, Operate, Manage, and Administrator, corresponding to level values 1 through 5, respectively.

B. Matter Development Model

At a very high-level, Matter is known to be an open-source standard with public protocol specifications and open-source implementation, as a project namely "connectedhomeip" [31] maintained on GitHub by its open-source community coordinated by CSA. Based on CSA and public information in the commits on GitHub, the open-source developers include those from Apple, Google, and other companies from IoT industry [18]. They contribute to the open-source layer of a Matter-enabled product and then individual vendors develop specific applications and features on top of the open-source layer of Matter (detailed below).

For an end-user facing IoT product supporting Matter, we decompose Matter implementation model into three layers:

• *L1: The Matter open-source layer*. By inspecting the Matter project on GitHub, the open-source implementation of Matter (L1) includes (1) device-side firmware for a range of IoT device types (e.g., smart plug, smart light) across multiple platforms such as ESP32 and ARM; (2) implementation of Matter controller protocol in the form of an SDK called the CHIP SDK [30]. The CHIP SDK implemented a range of primitive capabilities to communicate with and control the Matter device — Matter devices are any IoT devices whose firmware supports communication with Matter controllers through the Matter protocol. The CHIP SDK exposes the primitive capabilities through Application Programming Interfaces (APIs) illustrated in Appendix Table IV. These APIs are used by Matter controllers implemented by individual vendors to control Matter devices (see below). For example, the API `pairing code` with a secret pairing code as an argument can be used by Matter controllers to pair with the device; the API `accesscontrol write acl` can be used by Matter controllers (that is paired with the device and authorized) to update the access control list (ACL) maintained in Matter devices.

• *L2: Vendors' Matter application layer (local).* At L2, IoT vendors develop a few key components under the Matter context: (1) Matter controller (*MC*): The vendors' Matter controllers integrate the CHIP SDK and use the SDK APIs to bind with and control IoT devices, leveraging the primitive capabilities implemented by the CHIP SDK. Matter controllers are commonly implemented either as part of an IoT mobile app (e.g., Apple Home [10] and Google Home app [7]) or inside an IoT gateway hub (e.g., Apple HomePod, Google Nest Mini, Amazon Echo Dot [6]). (2) User-facing Matter control capabilities and interfaces (*UMCCI*): *UMCCI* is usually implemented in mobile apps for users to view, use, and control Matter devices through Matter controllers. Conceptually, Matter controllers are transparent to users. The Matter open-source project additionally comes with a command-line tool called *CHIP Tool*, which integrated the CHIP SDK and can be used by developers as a Matter controller to test connection and control with Matter devices.

• *L3: Vendors' cloud layer (remote).* At L3, although Matter is designed for local-control (§ V), by studying 8 IoT vendors, we notice that vendors tend to come with their IoT backend (cloud) service for remote or centralized user and device management. For example, when the Matter controller is in an IoT gateway device like Apple HomePod, Apple users (using the Apple Home app) can remotely send IoT commands to iCloud, which performs authorization. If the authorization is successful, the cloud will send the commands to the Matter controller local to the device. After that, the controller will go with the Matter protocol to operate the device. Such a remote access paradigm is a combination of Matter and the prior cloud-based IoT access [39], which is emergent along with wide adoption of Matter.

III. Systematic Security Analysis for Matter Development Model

Security risks in vendors' Matter integration. By studying Matter integration of 11 popular IoT devices from 8 vendors, we notice that although the CHIP SDK offers a set of standard APIs and primitive control capabilities under the Matter standard, how individual vendors leverage those APIs to design user-facing Matter control capabilities and interfaces (*UMCCI*) are different, up to individual vendors' interpretation of the Matter standard and the functionalities that individual vendors aim to offer.

Systematic security analysis for *UMCCI* flaws. To systematically analyze *UMCCI* flaws, denoting absence or mistakes in offering user-facing Matter control capabilities by vendors, we summarize security requirements and responsibilities expected from vendors in Matter integration. Based on the three-layer Matter development model, *UMCCI* flaws are user-facing aspects in the Matter design space and primarily concentrate on vendors' L2 development (Figure 2). Specifically, we propose the following security requirements ($P1$ to $P3$ below) in vendors' L2 development and analyze whether they are violated in vendors' user-facing IoT apps and controllers. We note that even issues of user-facing control capabilities stemming from L1 (Matter open-source implementation) and L3 (vendor cloud) (with examples in § IV-B) can be identified from analyzing non-fulfilled security requirements at L2 development, presenting a novel user-centric security analysis in our study.

• *View access-control information (P1)*: provide legitimate users with capabilities to view or query Matter access-control information, including all Matter controllers of the IoT devices, all Matter nodes (e.g., Matter devices, controllers) under a Matter fabric, and Matter ACL information maintained in Matter devices.

• *Access-control information trustworthiness (P2).* The access control information shown to users should be true and trustworthy.

• *Update access-control information (P3)*: provide capabilities for legitimate users to establish or revoke access-control for Matter devices.

For systematic analysis, we assessed whether vendors' Matter products violate each security requirement, assisted by *UMCCI* Checker in § VI: for $P1$-$P2$, we examine app UI to find whether access-control information is available and true given our attacks; for $P3$, we examine if the access-control capability (i.e., Matter binding and revoking) provided by real vendors is implemented and effective. A violation of the above security requirements indicates a *UMCCI* flaw.

In the following sections, we report our security analysis results, i.e., 6 types of *UMCCI* flaws we find in violation of the above security requirements by popular vendors. We categorize the *UMCCI* flaws based on whether they are mainly caused by the vendors in adopting Matter (§ IV) or the flaws are due to design weaknesses deep in the Matter standard itself potentially impacting all vendors (§ V).

Notably, in analyzing the vendors' Matter design and development, we have deployed 11 different Matter devices that work with the vendors' apps and controllers (Table II and Table III in Appendix). Based of Matter's interoperability nature, a vendor's Matter device usually works with other vendors' controllers (otherwise, it can be considered as a functionality bug, which sometimes happens but is out of our current research scope). In our experiments to verify the *UMCCI* flaws below, we extensively used a consumer Matter device (the SM-PW703 smart plug from Xenon, CSA Vendor ID 0x1481) which we referred to as device X. For each *UMCCI*, we confirmed the issues also using other Matter devices (Table III). We release all PoC videos and supporting materials on our website [13].

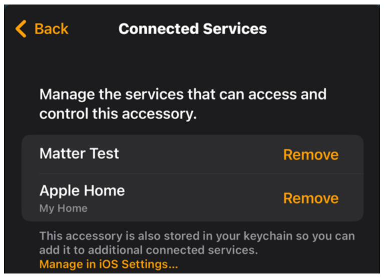
Threat Model. We consider the IoT infrastructure and systems to be benign (hardware and firmware in the devices, vendors' software, the cloud, networks, etc.). The adversary has access to public information from the Matter open-source project, including the Matter standard and protocol specification and open-source implementation. He can open user accounts with IoT vendors and can analyze network traffic between the IoT cloud, Matter-enabled IoT devices, and the vendor app under his control. He cannot eavesdrop on or interfere with the communication of other users' devices and apps. We also consider real-world access-control delegation and device-sharing scenarios (e.g., Airbnb/offices): the adversary (e.g., prior employees, guests or tenants — delegatee users) can have temporary access to Matter devices and their access rights are subject to revocation by device owners or administrators.

IV. Error Space in Vendor Design

This section reports three types of *UMCCI* flaws due to vendor mistakes (§ IV-A to § IV-C). The findings bring to light a key security challenge for IoT vendors to integrate the Matter protocol. That is, vendors generally failed to design and develop adequate security awareness and control capabilities in their user-facing Matter-compatible apps, especially when the user-facing security capabilities are not defined and standardized by the Matter protocol. Specifically, even most high-profile vendors such as Apple, Google, SmartThings failed to ensure that the Matter protocol's key semantic elements and operational states related to awareness, integrity and functions of access control should be identified and provided to users.

A. Flaw Type 1: Concealment of Matter Fabrics

Hidden Matter fabrics in Apple Home. Based on design of the Matter protocol, once users pair a Matter device X with both Apple Home and another controller such as CHIP Tool (through Matter pairing code), X exists in two separate Matter fabrics. Two controllers representing the two fabrics can be viewed on the "connected services" page in Apple Home (Figure 3), with each controller identified by Matter parameters such as Vendor ID and Label value. In Figure 3, "Matter Test" is the vendor name of CHIP Tool, based on an Matter-official mapping from Vendor ID to vendor name [9], [34]; "My Home" is the Matter Label value used by Apple Home controller. However, when we use the CHIP Tool controller to issue Matter's `operationalcredentials read fabric` command to query X's fabric list, we find that X is actually in three fabrics, identified by Vendor ID 4937 for Apple Home, 65521 for CHIP Tool, and an additional one 4996 for Apple Keychain (Figure 4). The last one is hidden by Apple Home to users. In our research, once we change the Vendor ID of the CHIP Tool controller to 4996 (either before or after pairing), as an attempt to masquerade as Apple Keychain, we find that the CHIP Tool is hidden in the "connected services" page of Apple Home. This reflects that Apple Home's logic for hiding the Apple Keychain fabric relies on whether the Vendor ID equals 4996, which is the Vendor ID for Apple Keychain officially assigned by Matter [9].



Manage the services that can access and control this accessory.

Matter Test	Remove
Apple Home My Home	Remove

This accessory is also stored in your keychain so you can add it to additional connected services.
Manage in iOS Settings...

Fig. 3: Connected Controllers for Device X in Apple Home

Such a "hiding fabric" design is likely for reducing confusion to normal users since Apple Keychain is a concept internal to Apple Home and HomeKit. On iOS, when any vendor's IoT app establishes pairing as a controller with a Matter device, required by Apple HomeKit's developer policy [11], [12], the vendor app must create a separate fabric for Keychain and related Matter credential (by calling the CHIP SDK API `Operationalcredentials`) and store the credential to iCloud keychain database by calling HomeKit API `Home.addAndSetupAccessories()` [2], [3], [22]. Behind the scene, Apple Home on iOS can leverage the Keychain fabric credential, acting as an independent Matter controller to the device, and use the Matter command `addnoc` to create the Apple Home fabric generating related credentials. From an end-user's perspective, even if she only pairs with the device using a third-party vendor app, she can alternatively use the Apple Home app to control the device as if it is already paired using Apple Home. This is done by simply clicking an icon button related to this device appearing in the Apple Home app (Appendix Figure 11) and correspondingly the Keychain fabric will create the Apple Home fabric. In such a "seamless" process designed by Apple, the Keychain fabric is more like a bridge, while the actual operation of the device relies on Apple Home and the vendor app's fabrics and controllers. Further, the Keychain fabric's credential (stored in iCloud database) is accessible to other Apple devices of the user and, thus, the user can use the Apple Home app on other devices to easily establish a Matter controller for the device without pairing again.

Hidden fabric attack. Such a vendor-level design by Apple is aimed at consumer usability, but it brings in practical security risks found in our study. Specifically, a malicious delegatee or guest user, once invited for at least temporary use of the Matter device (through a Matter pairing code), can exploit the hidden fabrics. Specifically, he can use a controller like CHIP Tool that is configured to bear the same Vendor ID as Apple Keychain. Such a fabric and controller are hidden in the owner's Apple Home app, and even when the owner removes the user (by removing any of his controllers shown in Apple Home), the hidden controller is unknown and invisible to the owner (see PoC exploit below).

```
Fabrics: 3 entries
[1]: {                 [2]: {                [3]: {
RootPublicKey: ###     RootPublicKey: ###    RootPublicKey:###
VendorID: 4937         VendorID: 4996        VendorID: 65521
FabricID: 3516426054   FabricID: 661177271   FabricID: 1
NodeID: 3650752681     NodeID: 3823095863    NodeID: 100
Label: My Home         Label:                Label:
FabricIndex: 63        FabricIndex: 64       FabricIndex: 62
}                      }                     }
```

Fig. 4: Listing Fabrics for Device X Using CHIP Tool

PoC exploit. A Matter device owner (victim) uses the Apple Home app to share the device with a guest, who can be malicious. To this end, the owner generates a Matter pairing code in Apple Home (Appendix Figure 10). The attacker initially pairs with the owner's device using a regular app that has implemented a Matter controller (such as the Smart Life app of Tuya in our experiment). Afterwards, the attacker shares the device with his CHIP Tool, which also acts as a Matter controller paired with the device. By configuring the Vendor ID 4996 in his CHIP Tool through the CHIP SDK API `commissioner-vendorid [Vendor_ID]`, the same as Apple Keychain during the pairing process, the attacker's CHIP Tool is invisible in the owner's Apple Home app, effectively establishing a covert control channel. In Apple's Home app, the owner can remove the guest user by removing his Matter controller of Smart Life in the app UI. Even if the owner removes the guest user, the attacker's CHIP Tool will still retain full control privileges.

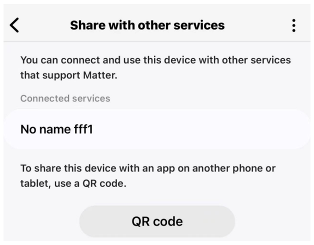
Stealthiness discussion. In real scenarios such as Airbnb and hotels, after the guest has checked out and the owner has removed him using the Apple Home app, the guest's CHIP

Tool remains completely invisible to the owner in Apple Home and has full Matter control over the device.

Matter control restoring attack. We find that the design of hidden Keychain fabric even enables non-technical savvy Apple users to easily attack Matter devices, with serious security and privacy implications to Matter device owners. As mentioned earlier, Apple Home, behind the scene, automatically leverages the hidden Keychain fabric as a controller to create the Apple Home fabric, such that the Home app is able to control the device without going through a Matter pairing process. Consider the logic process and scenario as follows. A device owner, as an Apple Home user, shares the device with a delegatee or guest user (e.g., Airbnb guest, employee) by sharing a Matter pairing code. The guest uses the Apple Home app to establish a Matter controller and use the device. Like any vendor app, the guest's Apple Home app silently creates a Keychain fabric, which, however, is hidden on the owner's Apple Home app. The natural way for the owner to revoke and remove the guest user is to go to "connected service" page of Apple Home app and remove the guest's fabric. Once this is done, the guest user's Keychain fabric is still effective, retaining full Matter control over the device, which is unknown to the owner based on Apple Home *UMCCI*.

PoC exploit. To easily exploit the leftover Keychain fabric, the guest user simply uses his Apple Home app, and clicks the icon button related to the device of the owner (Appendix Figure 11). Apple Home will re-recreate the Apple Home fabric using the Keychain fabric (see above), allowing the guest user to perform arbitrary Matter operations towards the device using the Apple Home app. Such a capability of the guest user is unknown to the owner and he can exploit it at any appropriate time. The consequence depends on the types of the devices, such as door locks that may incur serious security and privacy implications.

Similar "hidden fabric" flaws in Samsung SmartThings. A similar flaw was found by our *UMCCI* Checker when testing the Matter functionality of Samsung SmartThings. Unlike the flaw in Apple Home, the SmartThings app did not add an additional fabric but hid its own. This violates the security requirement $P1$ proposed in § III. Notably, the title of the page for querying connected services in the SmartThings app is "Share with other services", suggesting that the original intent of SmartThings' *UMCCI* design might have been to display fabrics other than SmartThings to the user.


Share with other services

You can connect and use this device with other services that support Matter.

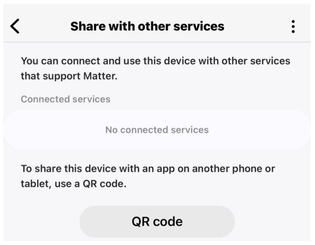
Connected services

No name fff1

To share this device with an app on another phone or tablet, use a QR code.

QR code

(a) When CHIP Tool with fabric label values "No name fff1"


Share with other services

You can connect and use this device with other services that support Matter.

Connected services

No connected services

To share this device with an app on another phone or tablet, use a QR code.

QR code

(b) When CHIP Tool with special label value "XsmartthingsX"

Fig. 5: Non-Display of SmartThings' Elements

We paired Matter device X separately with the SmartThings app and CHIP Tool. Subsequent queries of connected services, i.e., fabric information, using these two Matter controllers revealed that in the SmartThings app, only the fabric information of CHIP Tool was displayed, not the SmartThings app's own fabric (see Figure 5a). From CHIP Tool, the Vendor ID of the SmartThings fabric was seen as 4362, with the Label value being "SmartThings Hub XXXX" (where XXXX varies with the device and Samsung account).

We found that SmartThings obfuscated itself based on the controller's Label value and not Vendor ID, unlike Apple. Any fabric whose Label value includes the string "smartthings", case-insensitively matching the regular expression `/smartthings/i`, would be hidden in the SmartThings app's connected services list (see Figure 5b). This indicates that SmartThings uses flawed logic dependent on Label values to determine if a fabric is its own, creating a security vulnerability. Therefore, a guest acting as a malicious attacker could exploit this flaw in SmartThings to achieve a hidden fabric attack similar to those against Apple Home and Apple Keychain, namely establishing a covert control channel for the device without the owner's knowledge.

PoC exploit. Like the PoC with Apple Home above, a Matter device owner using SmartThings app shares the device pairing code with a delegatee user (e.g., a malicious guest). The guest can initially pair with the device using a normal app with a Matter controller (e.g., Smart Life in our experiment) and then share the device with his CHIP Tool that uses the crafted Label value `smartthings`. To configure Label value of CHIP Tool, the attacker uses CHIP SDK API `operationalcredentials update-fabric-label smartthings` (Appendix Table IV). The guest's CHIP Tool fabric is invisible in the SmartThings app, effectively establishing a covert control channel within the Matter protocol.

Stealthiness discussion. In real scenarios like Airbnb, this flaw allows the malicious guest's CHIP Tool to become invisible to the owner in the SmartThings app. Hence, even after the guest leaves, the owner can remove the guest's normal controller, but cannot identify or remove this hidden CHIP Tool controller in the SmartThings *UMCCI*, allowing the guest to retain covert control over the devices.

B. Flaw Type 2: Incomplete Access Control Displays in Intra-Fabric Matter Systems

Different from Apple Home, SmartThings, Google Home, etc., that are designed to create separate fabrics for delegatee users to control Matter devices, other vendors like Tuya place delegatee users into the same fabric as the owner. Interestingly, the former design requires the delegatee users to go through the Matter pairing process using a pairing code (e.g., Apple Home), and the latter design enables the vendor to more seamlessly integrate Matter into their existing IoT products. In the Tuya app, for example, the owner simply invites a delegatee user (using his email or Tuya account) to the Tuya home as a low-privilege guest user. Then the guest's Tuya app, equipped with a Matter controller implementation, is added to the owner's existing Matter fabric, with the assigned Matter node ID and related credentials, enabling it to control the Tuya device. In such an intra-fabric Matter system, essentially, the guest user's permissions are determined by how the on-device intra-fabric ACL is configured. Properly managing user permissions requires proper management and display of intra-fabric ACL, which we find are error-prone for vendors in the absence of security analysis like ours, presenting another type of *UMCCI* flaw.

Unknown over-privilege in Tuya fabric. Unlike Matter controllers of Apple Home and SmartThings that hold the root authority of the fabric (by holding the fabric's root certificate called RCAC, § II-A), the Tuya cloud holds the authority of Tuya home Matter fabric and offers an API `thing.m.matter.icac.issue` to sign ICAC when this is requested by Tuya apps. Specifically, once a guest user is added to a Tuya home, the guest's Tuya app generates a public key and private key pair and calls the Tuya cloud API to sign the key pair, obtaining an ICAC of the Tuya home fabric and Matter node ID. Here, only user accounts already added to the Tuya home are allowed to do so. Then the guest user's Tuya app is able to serve as a Matter controller and operate the device. A benign guest user's privilege in the Tuya home is highly restricted with only basic "view" and "operate" capability designated by Tuya home ("home" is a common concept in IoT products); she cannot, for example, remove the home owner, change the owner's privileges, or edit devices in the home. However, her actual control capabilities over Matter devices are determined by the on-device in-fabric ACL. In the case of Tuya device design, we find that her privilege is configured at level 5 ("Administrator", the highest). By sending arbitrary commands in Matter protocol directly to the Tuya device, her controller is empowered with full control capability. The problem is that, the actual privilege of Matter controllers determined by in-fabric ACL is not shown or known to the device owner, based on the *UMCCI* design of Tuya app. We show that this enables multiple attacks, elaborated below.

Covert control attack within the Matter fabric. The malicious guest can call the Tuya cloud API to generate an additional ICAC, used by another controller of his such as CHIP Tool to control the device. Such multiple controllers within the owner's fabric are not expected, and are not shown in the Tuya app *UMCCI*. Even after the owner removes the guest from the Tuya home, the CHIP Tool controller retains administrative level control over the device, unknown by the victim owner. Alternative to this "extra controller" attack, the guest can add malicious devices to the smart home invisible to the owner in the *UMCCI*, to operate the victim device (§ V-B).

PoC exploit. The device owner invites a guest (malicious) into his Tuya home and grants the guest level privilege who can only operate the device but cannot edit the device or other users. Through reverse engineering the Tuya app, the guest can hook his app to interact with the Tuya cloud through some APIs. The guest can initiate a request to the clouds' `thing.m.matter.icac.issue` API, with arguments including the Tuya home's group ID, and an ECC public key to be signed (Appendix Figure 12a). The response includes RCAC and ICAC certificates signed by the Tuya cloud, the fabric ID of the Tuya home, and the node ID assigned to the guest within that fabric (Appendix Figure 12b). Additionally, by requesting the cloud API `thing.m.matter.device.node.batch.get`, the node IDs of devices within the fabric in the home can be obtained. The guest installs the ICAC to CHIP Tool by modifying the ini configuration file, further configures CHIP Tool's control parameters, including the commissioner's name, node ID, and Vendor ID (Table IV), and then can use CHIP Tool to control the device as the Tuya home's fabric administrator. The device owner then revokes the guest's access by removing him from Tuya home. Afterwards, the guest's CHIP Tool can still perform arbitrary operations on the Tuya device by issuing Matter commands such as `write acl` (Table IV).

```
ACL: 1 entries
[1]: {
Privilege: 5
AuthMode: 2
Subjects: 3 entries
  [1]: ###
  [2]: 65536
  [3]: 65537
Targets: null
FabricIndex: 1}
```

(a) Original

```
ACL: 2 entries
[1]: {                  [2]: {
Privilege: 5              Privilege: 1
AuthMode: 2               AuthMode: 2
Subjects: 2 entries       Subjects: 1 entries
  [1]: ###                  [1]: 65536
  [2]: 65537
Targets: null           Targets: null
FabricIndex: 1          FabricIndex: 1
}                       }
```

(b) Malicious ACL Data Altered by an Attacker

Fig. 6: ACL Data within the Fabric in Tuya Home

Permission downgrading attack within the Matter fabric. The malicious guest bears high privileges (based on ICAC) for the Matter device. Using the ICAC, his controller (a CHIP Tool in our PoC implementation) is able to call the CHIP SDK API `read acl` and `write acl` to read and update in-fabric ACL of the device. In our experiment, he downgrades the privilege of the victim owner's Tuya controller (node ID 65535) from 5 to 1 (Figure 6a and 6b) while keeping privilege of his own controller (node ID 65537) as 5. After such an attack, the owner is not even permitted to operate the device, even though the Tuya app *UMCCI* still shows she is the home owner (Appendix Figure 13), but never shows her true privilege over the Matter device defined by intra-fabric ACL.

PoC exploit. By extending the above PoC exploit, the guest uses his CHIP Tool to modify the device's ACL using Matter's `write acl [ACL_VALUE]` command, decreasing the privilege level of the owner from `5-Administer` to `1-View` (Figure 6b). The owner using his Tuya app can not control the device at all, nor can he learn the current privilege level of his controller for the device from the Tuya app.

Stealthiness discussion. In real scenarios like Airbnb or hotels, Tuya's permission structure erroneously grants guests elevated ACL permissions, enabling them to exploit hidden APIs and gain excessive control. The owner cannot view or manage ACL permissions for all members in the Tuya home, making it easy for malicious guests to use the CHIP Tool to establish an invisible Matter controller within the same fabric. Even after being removed by the owner, guests retain covert control over devices, leaving the owner unaware. This flaw further enables guests to launch permission downgrading attacks, where they can reduce the owner's ACL permissions to prevent normal device control—without the owner knowing the cause or source of the malfunction.

C. Flaw Type 3: Errors in Managing Connected Services

Inability to view connected Matter services in Amazon Alexa. Amazon Alexa now supports the Matter protocol and can normally complete pairing, control, and sharing operations with Matter devices. However, it lacks important functionality as a Matter controller, specifically the ability to view paired controllers or fabric information. This violates the security requirement $P1$ proposed in § III. We find that after pairing a Matter device with Amazon Alexa and CHIP Tool, there is no *UMCCI* within Amazon Alexa for querying fabric information, hence missing visibility of any other Matter controllers. If a device owner shares their Matter device with a guest (malicious) through Amazon Alexa, the owner has no way to revoke the guest's Matter control permissions other than by factory resetting the device. They also cannot prevent the guest from reopening the device's commissioning window and silently sharing the pairing code again (while the guest has permissions). The absence of fundamental features comes with serious security implications.

PoC exploit. A Matter device owner who uses Amazon Alexa to manage Matter devices shares a Matter device's pairing code with a guest, who can be malicious. The sharing code is generated in the Alexa app by the owner. The guest can arbitrarily choose apps that implement a Matter controller and possess permissions that are difficult to revoke because their presence is not visible in the owner's Amazon Alexa app or *UMCCI*, effectively establishing a covert control channel within the Matter protocol.

Stealthiness discussion. In real scenarios such as Airbnb and hotels where the owner uses Amazon Alexa to manage Matter devices, as long as the guest is shared with the device and established a Matter controller, the controller is invisible to the device owner based on Alexa *UMCCI*, and cannot be revoked. Even if the guest shows to the owner while he deletes his Matter controller, he could have created additional Matter controllers for the device while he still has legitimate permissions, and these controllers are invisible to the owner based on Alexa *UMCCI*.

Incomplete hardware reset logic in Signify WiZ. Typically, with a hardware-level factory reset, one expects that any prior users or configurations of the device, if they exist, are completely removed. However, we find that this is not the case for Signify WiZ [17]'s Matter devices and this cannot remove its established fabrics, intra-fabric ACL and control capabilities of established controllers, which violates normal security exceptions and our security requirement $P3$ (§ III). Interestingly, a software-level factory reset performed using the Signify WiZ mobile app by the device user (who has paired with the device) successfully factory resets the Matter protocols in the device. Further, Signify WiZ devices support prior control channels besides Matter, such as control by Google Home and Amazon Alexa with their smart speaker protocols and Signify WiZ app's proprietary protocol [40], [55] (not Matter protocols). Surprisingly, a hardware factory reset on Signify WiZ devices successfully resets those other control channels, but silently keeps the control capability by established Matter controllers. This vulnerability partially arises from Signify WiZ enabling already deployed devices to support the Matter protocol via Over-The-Air (OTA) updates, without timely updates to the logic for hardware resets. § V-C further discusses that the Matter protocol itself lacks a function to easily and completely reset in-device Matter status including fabrics and ACL.

PoC exploit. The owner shares the WiZ device with a guest (malicious). He can either use his WiZ app to generate a Matter pairing code and share with the guest, or let the guest freely configure and use the device. The guest uses any Matter controller app to pair with the device and establish control. Later, the owner performs hardware level factory reset through repeated switching aiming to remove the guest's access rights. However, the guest's Matter control to the device cannot be removed, which is unexpected by the owner. Below, we further discuss that the owner's WiZ app *UMCCI* cannot show the Matter controller of the device, like the Alexa attack above.

Stealthiness discussion. In real scenarios such as Airbnb and apartments, owners may share WiZ devices with guests (or tenants). After guests leave, owners can attempt to clear guests' control by a hardware reset. However, owners cannot realize that this reset method does not remove the guest's control over the devices through the Matter protocol, but other control channels of the guest such as Alexa and WiZ app are reset.

Inability to view Matter controllers in Signify WiZ app. The Signify WiZ app lacks the functionality to show the device's Matter fabrics and even WiZ app itself cannot serve as a Matter controller at all. Initially, Signify WiZ did not support the Matter protocol. To accelerate the adaptation of its products for the Matter protocol, Signify WiZ not only pre-installs Matter capabilities in new products, but also offers OTA updates for deployed older models to support Matter. When users of Signify WiZ products wish to utilize the Matter protocol, Signify WiZ requires them to first pair and connect their devices with the WiZ app (not through Matter, but using WiZ's original, proprietary protocol). Subsequently, the WiZ app allows the activation of the device's Matter commissioning window and the retrieval of Matter pairing code. Users can use any app that implements Matter controller, with the pairing code, to pair with the WiZ device. The WiZ app itself, version 1.17.3 at the time of our research, does not support Matter pairing. Nor is the WiZ app of the owner able to query and show Matter controller established for the WiZ device, a violation of security requirement P1 if the owner relies on the WiZ app to manage the devices.

PoC exploit. A device owner who uses the WiZ app shares her WiZ device via the Matter protocol with a guest (malicious). Once the guest uses Matter to establish a controller for the device, it becomes difficult for the owner to identify the device's access control status within the Matter protocol, as the Signify WiZ app does not provide the capability to query the device's fabric or controllers. Nor can the owner remove the guest's Matter controller and fabric using the WiZ app. This limitation can inadvertently allow the guest to maintain stealthy control over the device.

Stealthiness discussion. In real scenarios such as Airbnb and apartments, WiZ device owners who rely on the WiZ app and hardware reset capability are unable to properly view, identify or remove the guests' Matter control over the WiZ devices. This easily allows guests to retain control over the devices without the owner's knowledge.

V. Error Space in Matter Standard Design

This section reports three types of *UMCCI* flaws due to insufficient or flawed security design in the Matter protocol itself. The flaws and related attacks are initially found and launched in popular vendors' IoT apps, while our root cause analysis shows that the problems are caused by flawed design of Matter, making critical access control information and operations (e.g., resetting) related to Matter devices hard to verify, gather and perform in vendors' Matter implementation. Our findings call for immediate and thorough security analysis of the Matter protocol itself.

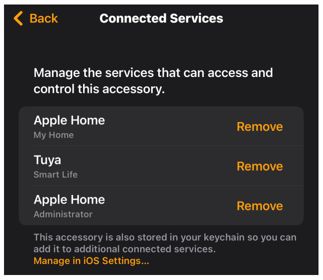
A. Flaw Type 4: Untrusted Fabric Information

Matter Vendor ID and Label forgery attack in Apple Home. The Matter protocol design does not verify parameters used by Matter controllers, such as Vendor ID and Label value. Consequently, Matter controller information of a device that can be shown to benign users is subject to forgery by malicious users, who, for example, use a controller implementation like CHIP Tool that can be configured to use arbitrary parameters. This makes it difficult for benign users to distinguish between legitimate and malicious or unauthorized Matter controllers. It can easily result in failure of controller revocation and the persistence of unauthorized controllers that should be removed, violating the security requirement $P2$ (§ III).

When Apple Home as a Matter controller is used by device owner or administrator, it provides a feature to query and view the Matter controllers paired with the device, on a "connected services" page of the app (Figure 7a). Based on Apple Home *UMCCI* design, the page lists basic information of all fabrics/controllers associated with the device, including the Label value of each controller and the vendor name, where the vendor name is mapped from the Vendor ID used by the controller to the corresponding vendor name (based on the Matter-official mapping [9]).

Consider a realistic scenario where the device owner uses the Apple Home app and shares the device temporarily with a guest user (e.g., Airbnb guest, tenants, employees) through a Matter pairing code. The guest can be malicious aiming to gain unauthorized control even after his permission is revoked. When using a Matter controller to control the owner's device, he can use fake Vendor ID and Label value. Normally, the guest may use a controller implementation like the Tuya app, which bears Vendor ID 4701 and Label value "Smart Life", so the owner easily knows that controller belongs to the guest in the Apple Home *UMCCI* (Figure 7a) and can correctly removes it when the guest leaves. However, a malicious guest can use a crafted controller like the CHIP Tool being configured with the Vendor ID 4937 (for Apple Home) and the Label like "Owner" or "Administrator" resembling a controller of the true owner (the last one listed in Figure 7a). Actually, the true owner's Apple Home fabric (the first one listed in Figure 7a) bears the same Vendor name but uses a Label value "My Home" (implemented by Apple Home). This can practically confuse the true owner in deciding which controller belongs to the guest and should be removed when he leaves. Once the owner removes the wrong controller, the guest retains control capabilities, violating the security expectations of the owner. Notably, Apple has acknowledged the problem and is making new design to mitigate the problem, which is non-trivial without a standard, assured verification mechanism at the Matter protocol level.

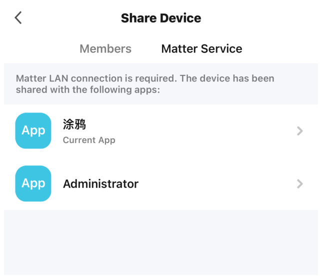
PoC exploit. The Apple owner generates a new Matter pairing code of the device and shares it with the guest (malicious). The


Back
Connected Services
Manage the services that can access and control this accessory.

Apple Home My Home	Remove
Tuya Smart Life	Remove
Apple Home Administrator	Remove

This accessory is also stored in your keychain so you can add it to additional connected services.
Manage in iOS Settings...

(a) In Apple Home


Share Device
Members Matter Service
Matter LAN connection is required. The device has been shared with the following apps:
App 涂鸦
Current App
App Administrator

(b) In Tuya

Fig. 7: Falsifying Vendor ID and Label Values

guest pairs with the device using any app that has implemented a Matter controller (the Smart Life app of Tuya in our experiment). Afterwards, the attacker shares the device with his CHIP Tool, which also acts as a Matter controller paired with the device. The CHIP tool uses a Vendor ID 4937 and Label value "Administrator". After the owner removes the guest user by removing his Smart Life controller in the Apple Home *UMCCI*, it is difficult for the owner to associate the malicious guest's CHIP Tool controller with the guest. In the implementation, we configure the Vendor ID of CHIP Tool with the command `commissioner-vendorid [Vendor_ID]`, and configure the Label value with the `update-fabric-label [Label_Value]` command (Appendix Table IV).

Stealthiness discussion. In real scenarios such as Airbnb, device owners may use Apple Home to manage their Mater devices, share devices with temporary guests, and remove guest users after they check out, expecting that the guest user's permissions are transparent when they stay and are completely revoked after they leave. With our exploit above, the malicious guest can leverage fake Matter Vendor ID and Label value to easily confuse benign owner, making it difficult for the owner to correctly identify and completely remove the guest user's Matter controllers (the Smart Life and the CHIP Tool). The consequence is that the guest retains full Matter control over the owner's device even after the owner removes the guest (by removing the obvious Smart Life of the guest, Figure 7a).

Similar flaws in Tuya. From the description of the flaw in Apple Home above, it is evident that the *UMCCI* display flaws encountered by vendors when using the Matter protocol predominantly occur on the interface for querying connected fabric information, and the exploitation of these vulnerabilities often hinges on the Vendor ID and Label values. A similar flaw was found by our *UMCCI* Checker in the Matter functionality of Tuya. The display logic for Vendor ID and Label information in the Tuya app is disorganized and does not correctly present the information in a complete and accurate manner within the *UMCCI*. This violates the security requirement $P2$ proposed in § III. Utilizing this flaw in the Tuya app can achieve an attack effect similar to that observed in Apple Home (see Figure 7b). According to the Tuya app's code logic got through reverse engineering, Tuya first checks if the fabric's Vendor ID matches any of the following: 4142 (LG ThinQ), 4321 (noted in the code as SmartThings), 4631 (Amazon Alexa), 4937 (Apple Home), 4996 (Apple Keychain), 24582 (Google Home). If it matches one of these Vendor IDs, the Tuya app displays the vendor name. If not, it then checks if the Label is null; if null, the Tuya app displays "Third-Party App";

if not, it displays the specific content of the Label. Furthermore, Tuya uses the Matter protocol's `isCurrentFabric` function to determine if it's Tuya fabric, marking it additionally as "Current App" if true. The analysis of the code revealed that when the controller's Vendor ID does not match any listed in the code, the text output in the Tuya app becomes the customizable Label value. At this point, similar attack strategies to Apple Home can be utilized, using CHIP Tool to display strings such as "Administrator" on the connected services of *UMCCI*, confusing the device owner to prevent them from revoking the attacker's control over the device (see Figure 7b).

PoC exploit. We implemented proof-of-concept attacks using our device X. A device owner using Tuya app shares the device's Matter pairing code with a guest who is a malicious attacker. The guest can then pair with the owner's device using CHIP Tools with a crafted Vendor ID that is not one of the following: 4142, 4321, 4631, 4937, 4996, or 24582, and a fabricated Label value. The owner will only be able to see the malicious Label values fabricated by the guest in the Tuya app (see Figure 7b). Consequently, similar to the previously mentioned Apple Home flaws in § V-A, the owner may struggle to distinguish the authentic fabric entries from the forged ones in the list of connected services.

Stealthiness discussion. Similar to the stealthiness of Apple Home flaws in § V-A, in real scenarios such as Airbnb, when owners use the Tuya app to share devices with guests via the Matter protocol, guests can also use the CHIP Tool to alter the Vendor ID and Label values, creating deceptive fabrics that confuse the owner. Additionally, the Tuya app conceals the vendor name when a Label value is present, so the owner might only see a fabric labeled simply as 'Administrator' making it even more challenging to discern.

B. Flaw Type 5: Missing Capability for Intra-Fabric Node Display and Auditing

Invisible node and malicious device in Matter fabric. The Matter protocol does not provide functionality for the Matter controller to gather or query information about nodes already added to a fabric. Notably, a node is conceptually a Matter device or controller under a Matter fabric, and holds a credential (called NOC) signed and recognized by the fabric (§ II-A). If the node ID of a device is forgotten, it becomes irretrievable, and the Matter controller is unable to find out nodes in the fabric or issue control operations to the device, based on Matter protocol design (Matter version 1.3, § II-A). We find that some vendors have designed proprietary mechanisms to record node information of paired devices when designing Matter controllers. For example, the vendors' controllers record node information (to the cloud) when the device is paired or added to the fabric, and designed lists and icons in the *UMCCI* to display Matter devices that users have paired. However, once a malicious device joins the fabric without being recorded by the vendor's controller or cloud, its presence in the fabric and the capabilities it has are unknown to the benign controller, which essentially violates the security requirement $P1$ (§ III).

Specifically, building on the aforementioned attack process in § IV-B, the attacker can engage in further attacks. Based on the design of Tuya's Matter integration, every user added into a home by the owner (e.g., a guest user) is assigned a Matter controller node within the fabric of the home owner, while simultaneously adding the guest's Tuya account information to the Tuya cloud for that home. In benign scenarios, every Matter device added to the home (using the Tuya app) is allocated node information within the fabric and synchronized with the Tuya cloud. Normally, this information is available and can be displayed in the owner's Tuya app *UMCCI*. The Tuya app retrieves home and device information of the fabric directly from the cloud, but it is unable to gather such access control information from the local Matter networks and devices based on Matter protocol design. Hence, based on the aforementioned Flaw Type 2 in Tuya, the malicious guest obtained operational permissions for the fabric, and through his CHIP Tool controller, any subsequent actions by the attacker directly in the fabric can bypass the Tuya cloud. Next, the attacker can add any malicious devices to the fabric without the owner's knowledge, and these additions and malicious device are all invisible in the owner's Tuya app.

We find that a malicious, invisible Matter device in the owner's Matter fabric comes with serious security and privacy implications. Such a device could control other Matter devices within the fabric leveraging Matter's "devices control devices" functionality. Similar to a regular Matter controller, a malicious device in the fabric can send regular Matter commands to the victim device as long as the intra-fabric ACL in the victim device allows the malicious device node to do so. In this case, since the malicious guest obtained a controller, he is able to write ACL in the victim device using Matter command `write acl [ACL_VALUE]`, practically enabling the attack (see our PoC exploit implementation below).

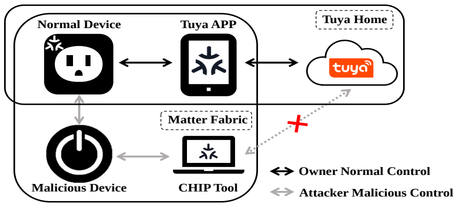


Fig. 8: Diagram of a Malicious Device Attack in Tuya

PoC exploit. By extending the Tuya convert control PoC (§ IV-B), a malicious guest can add a malicious device (e.g., a switch) to the owner's fabric. The guest binds the malicious switch with the benign device using Matter protocol cluster `Bind`, such that when the switch is pressed, the switch sends a Matter command `onoff` [32] to toggle the victim device. The guest also updates infra-fabric ACL of the victim device using the Matter cluster `AccessControl`, granting the switch the highest privilege level. Now the guest can operate the victim device by physically operating the malicious switch.

Stealthiness discussion. In real scenarios such as Airbnb, apartments or workplaces, the delegatee users such as guests, tenants, or employees can launch such attacks. The Tuya app *UMCCI* cannot show there exists a malicious device in the fabric or in the owner's Tuya home. Even after the owner removes the guest's Tuya account from the home or removes his Matter controller, the guest can still use such a hidden malicious device nearby to operate the owner's Matter device through the Matter protocol.

C. Flaw Type 6: Insufficient Definitions of Usage Rules for New Features

In § IV-C, we discuss two *UMCCI* flaws of Signify WiZ: incomplete hardware reset logic and the inability to view Matter controller information. These security flaws, however, are not solely the vendor's responsibility. The Matter protocol does not provide a function to completely clear in-device fabrics and reset its access control list, for example, at the invocation of a Matter API or press of a physical button on Matter device. This violates the security requirements $P3$ in § III. As a new protocol introduces new features, the designers and developers of the Matter protocol should provide more comprehensive security functions.

VI. Automatic Detection of UMCCI Flaws

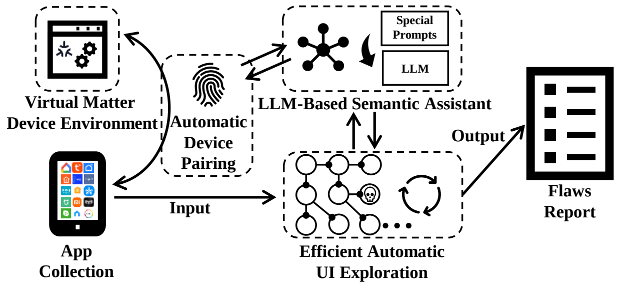

Virtual Matter Device Environment
Automatic Device Pairing
Special Prompts
LLM
LLM-Based Semantic Assistant
Output
Flaws Report
Input
App Collection
Efficient Automatic UI Exploration

Fig. 9: Architecture of *UMCCI* Checker

To help detect *UMCCI* flaws from real vendors' Matter integration, we developed *UMCCI* Checker that can automatically explore GUI elements within IoT apps and analyze the display and Matter control functionality provided in the apps (§ VI-A). Consequently, it identifies potential defects and detects IoT apps that are susceptible to *UMCCI* attacks, pinpointing *UMCCI* flaws in the apps for vendors to repair.

A. Design and Implementation.

Design overview. *UMCCI* Checker takes the IoT vendors' mobile apps as input, automatically executes and analyzes them, and reports *UMCCI* flaws found in the apps. The core idea of *UMCCI* Checker is to automatically search for and analyze *UMCCI* related GUI pages in the apps, assisted by large language models (LLMs), and verify whether the app meets security requirements $P1$ to $P3$ (§ III) providing users with necessary Matter control capabilities and displaying true, necessary access control information (e.g., Matter controllers, vendor IDs, fabric labels, and ACLs). The architecture of *UMCCI* Checker is outlined in Figure 9. Specifically, *UMCCI* Checker first prepares a Virtual Matter Device Environment (VMDE), which comes with a generic virtual Matter device that can be paired with any Matter-compatible IoT apps under test. VMDE eliminates the need for deploying physical devices in testing various vendors' apps, increasing automation and scalability. Then, the Automatic Device Pairing (ADP) module automatically pairs the virtual Matter device with the app under test, assisted by a module called LLM-based Matter Agent (L-MA). Also assisted by L-MA, the Efficient Automatic UI Exploration (EAUE) module searches for all *UMCCI* pages in the app, which are then checked by the Flaw Analysis and Reporting (FAR) module to identify any *UMCCI* flaws and produce patch suggestions for vendors. These key modules are elaborated below.

Virtual Matter Device Environment (VMDE). *UMCCI* Checker employed the open-source framework Matter.js [47], which is part of the Matter open source project and can be used to simulate real Matter devices. It is a TypeScript/JavaScript implementation of the Matter protocol of the device side and is originally for functionality testing during Matter CHIP SDK development. Also in VMDE is a CHIP Tool, which is used as a Matter controller and VMDE can automatically leverage CHIP Tool to pair with and control the virtual Matter device (like a real device under factory setting, Matter.js generates an initial Matter pairing code when it first runs). VMDE can dynamically configure parameters (Vendor ID and Label value) of the CHIP Tool and in-fabric access control list in the virtual devices for testing *UMCCI* flaws (detailed in FAR below).

Automatic Device Pairing (ADP). ADP automatically pairs the IoT app with the virtual Matter device. Starting from the app launching page, ADP queries LLM to identify the next UI operation necessary to navigate to the Matter binding page and performs the device binding. To query LLM, ADP first leverages our Pairing Prompt template (Appendix § A) and uses a set of information including a new pairing code of the virtual Matter device (obtained by CHIP Tool being a connected controller), the list of UI elements on the app's launching screen, and the app screenshot to populate the Pairing Prompt template, generating a specific pairing prompt; then ADP invokes the L-MA along with the pairing prompt. L-MA queries the external LLM and returns LLM's output to ADP. Specifically, based on the Pairing Prompt, the LLM outputs an ordered list of UI elements suggesting ADP to operate on next for the pairing goal. ADP then automatically performs UI operation. In implementation, ADP employs Appium [21], a generic UI automation development framework, to extract UI elements from the app pages and automatically perform the intended UI operation. *UMCCI* Checker keeps leveraging the latest UI states (UI elements, screenshot) to query LLM to be guided for the next UI operation, essentially performing a depth-first search along a potentially shortest path in navigating through the app GUI, until the virtual device is successfully paired with the app. Specifically, ADP's pairing attempts stop when (1) the output of LLM indicates success of pairing or (2) the pairing cannot succeed within 5 minutes for the app.

Efficient Automatic UI Exploration (EAUE). *UMCCI* Checker automatically explores and identifies all *UMCCI* pages in the app. The exploration algorithm employs a depth-first search (DFS) approach to automatically navigate through the app's GUI, and meanwhile leverages LLM to help navigate to and find *UMCCI* pages efficiently (as early as possible). More specifically, *UMCCI* Checker generates a directed graph G, where explored page states serve as nodes, and GUI operations leading to state transitions (e.g., clicks) represent edges. Specifically, a node comprises all UI elements extracted from the current app page denoted as a set S_e, which includes UI elements along with their attributes (such as class, text, and element ID). In the automatic GUI exploration, EAUE leverages LLM to help identify and prioritize the next UI operation that may most likely lead to *UMCCI* pages. Specifically, EAUE uses UI information recorded in the current node and the page's screenshot to populate our Searching Prompt (Appendix § A) and invokes L-MA, which queries the employed LLM. Based on the prompt, LLM returns two pieces of information: (1) whether the current page is a *UMCCI* page of interest (based on semantics similar to “Connected Services” and “Matter controllers”), and (2) the ordered list of UI elements on the page to operate on to further look for and explore *UMCCI* pages. Based on the LLM guidance, EAUE records if the page is a *UMCCI* page, and continues to operate on the next UI element, possibly transitioning to the next node on the graph G (i.e, a new app page or the same page with different states of the UI elements). EAUE compares each newly visited state with those already in G to avoid creating redundant nodes.

After exploring a branch of new pages or nodes, the exploration backtracks to the previous page or node and continues to explore other unprocessed UI elements (those also suggested by LLM but lower on the ordered list). This ensures comprehensive coverage of all interactive pathways within the app until all *UMCCI* pages are analyzed. Pseudocode of the DFS exploration Algorithm 1 is on our website [13].

Flaw Analysis and Reporting (FAR). FAR generates a comprehensive report detailing the identified flaws, including screenshots of the analyzed *UMCCI* page(s). Specifically, FAR uses node information of the *UMCCI* pages along with the screenshots of these pages to populate a Checking Prompt template (Appendix § A) and invokes L-MA, which queries LLM to check *UMCCI* flaws. FAR includes multiple rounds of testing considering the CHIP Tool as a (malicious) Matter controller: in each round, the CHIP Tool uses a different pair of Vendor ID and Label value, either copied from the app’s Matter controller (obtained by the CHIP Tool reading the virtual device’s fabric) or randomly generated. This is to test whether the app *UMCCI* pages properly display the CHIP Tool controller. In an additional testing round, VMDE programmatically changes Matter ACL in the virtual device for the vendor app’s fabric by lowering the app controller’s privilege from the highest level “Administrator” to the lowest level “View” (see fabric ACL in § II-A). This round tests whether the app *UMCCI* displays true privileges of the app controller for the device based on in-fabric ACL.

LLM-based Matter Agent (L-MA). LLM-based Matter Agent are empowered with three specialized prompt templates to perform key functions: assisting in decision-making for UI element interaction during device pairing (Pairing Prompt), *UMCCI* page search (Searching Prompt), as well as verifying security violations on *UMCCI* pages (Checking Prompt), used by ADP, EAUE, and FAR respectively to generate specific prompts (see above). LLM-based Matter Agent employs ChatGPT 3.5 and 4 as the LLM, can be easily configured to employ other LLMs. Detailed prompt templates are provided in Appendix § A.

Technical challenges in UI automation and our solutions. We detail several technical challenges in app UI automation and our solutions as follows.

C_1 *How to eliminate the redundancy brought by stacked elements?* In some cases, elements on a page may overlap, and clicking on any of them leads to the same new page. These overlapping elements are equivalent in terms of page transitions, and their redundancy can contribute to state explosion.

S_1 *UMCCI* Checker first filters all elements on the page that satisfy `clickable="true"` to identify clickable elements. When selecting an element e to generate touch input, *UMCCI* Checker calculates and filters out all other elements covered by e based on their positions in the UI page, thereby pruning unnecessary branches (Algorithm 2 on our website [13]).

C_2 *How to tackle dynamic UI elements?* Specifically, when *UMCCI* Checker touches a UI element e on page n, the application navigates to a new page. After thoroughly exploring the new page, it generates the back event to backtrack and explore the remaining elements on page n. However, the app may not be able to return to n since some page elements can disappear after the event e occurs unless one restarts the app.

S_2 During DFS backtracking, *UMCCI* Checker identifies the corresponding node n in the directed graph G, compares the element set of that node with the element set on the current app page, and identifies the disappearing elements S_{de} by the difference between the two sets. By appending each of S_{de} to the path from the initial state to n, *UMCCI* Checker determines all the paths to the lost states, facilitating quick access when restarting the app.

C_3 *How to ensure safe navigation and interaction? UMCCI* Checker may trigger irreversible operations, such as removing the devices, or unregistering the account, which prevents further analysis, even restarting the app can not help.

S_3 This challenge is addressed by implementing a smart blacklist mechanism that identifies and filters elements related to hazardous irreversible operations by gathering comprehensive descriptions of the elements and querying LLM-based Semantic Assistant. The descriptions include the text within the elements’ attributes and their screenshots. This strategy ensures that *UMCCI* Checker does not inadvertently alter the application’s state, thereby preserving its intended functionality and stability.

Other implementation details. We implemented *UMCCI* Checker with 253 lines of Python code and 844 words LLM prompts, based on the UI automation framework Appium [21] and the LLM ChatGPT 3.5 and 4 [27]. When testing apps, we use driver UI Automator [20] for Android and XCUITest [1] for iOS. We will release the full source code of *UMCCI* Checker upon acceptance of the paper.

B. Example of UMCCI Checker Analysis.

We use *UMCCI* Checker to perform a *UMCCI* security analysis on the SmartThings app (v1.8.14.26), demonstrating the step-by-step walkthroughs of *UMCCI* Checker.

• Step 1: Initializing the VMDE. *UMCCI* Checker initiates the virtual device and retrieves an initial Matter pairing code from it. Then *UMCCI* Checker initiates the CHIP Tool, which pairs with the virtual device using the pairing code.

• Step 2: Initializing the SmartThings mobile app. *UMCCI* Checker installs and launches the SmartThings app on the connected phone (Xiaomi Mi10 running Android 13). Then we manually complete the SmartThings account registration and login and thus the app is ready for *UMCCI* testing.

• Step 3: ADP. *UMCCI* Checker populates the Pairing Prompt template as input to the L-MA. *UMCCI* Checker then automatically performs ADP operation in the SmartThings app until the virtual device is successfully paired with the SmartThings app (see screenshots and UI operation sequence on our website [13]). Appium employed in *UMCCI* Checker is configured to automatically grant all the permissions when a permission dialog pops up.

• Step 4: EAUE explores and identifies all *UMCCI* pages in the SmartThings app. EAUE constructs the Searching Prompt template as input to the L-MA. In the SmartThings app, *UMCCI* Checker identified one *UMCCI* page titled "Share with other services" (see the figures on our website [13]).

• Step 5: FAR. *UMCCI* Checker analyzes all *UMCCI* pages with multiple rounds of testing based on constructing the Checking Prompt template as input for L-MA. In the SmartThings app, when the Label value of the CHIP Tool is set to match the SmartThings app's Matter controller, specifically match the /smartthings/i regular expression, the CHIP Tool's Matter controller is not shown on SmartThings app's *UMCCI* page, indicating the presence of Flaw Type 1 mentioned in § IV-A (see the figures on our website [13]).

C. Evaluation

By analyzing 10 IoT apps of 9 vendors, *UMCCI* Checker successfully identified 13 *UMCCI* flaws (Table II), all confirmed on our real devices, with multiple flaw types and novel attacks elaborated in § IV and § V. *UMCCI* Checker primarily focused on automatically identifying Flaw Type 1 to 4. Automatically testing Flaw Type 5 involves adding a "malicious" node to the app's fabric, and this requires signing a NOC for the node using the app fabric's certificate (RCAC or ICAC, see § II-A), which we do not have. Flaw Type 6 is related to identifying the missing security function in the Matter specification (i.e., a function to reset in-device Matter fabric and access control), which is out of *UMCCI* Checker's current scope. Our initial experiment shows that this can be achieved using LLM or natural language processing (NLP) to analyze API documents of the CHIP SDK and shows that there is no such API to reset Matter control for Matter devices. Based on CVE [8] and public information, no known *UMCCI* flaws in the 10 apps were missed by *UMCCI* Checker.

Performance overhead. We evaluated *UMCCI* Checker on all 10 IoT apps using a laptop with an AMD Ryzen 7 6800H CPU and 16GB of memory, and a Xiaomi Mi10 smartphone. Based on average results from five rounds of testing, *UMCCI* Checker took 194.58 seconds and utilized 20,320 KB of memory to discover all *UMCCI* pages within one app. We set a limit of five minutes to run the EAUE module for testing each app. The average total time for testing each app is 263.50 seconds.

Comparison with state-of-the-art tools. We compared two prior app UI automation tools DroidBot [44] and AutoDroid [54] with *UMCCI* Checker. Since AutoDroid and DroidBot are not designed for flaw analysis and mitigation, we compared them with *UMCCI* Checker on two key UI tasks on the 10 IoT apps: (T1) automatic pairing IoT apps with Matter devices; (T2) searching for the Matter "Connected Services" page in the app. *UMCCI* Checker succeeded in all 10 apps, while AutoDroid failed in 7 (70% failure). DroidBot, without semantic analysis capabilities, was not applicable for such tasks at all. The results are detailed in Table I.

AutoDroid, being Android-only, could not test Apple Home, whereas *UMCCI* Checker is cross-platform. When AutoDroid was tested on other apps like SmartThings and Amazon Alexa, it faced challenges C_1 to C_3 and failed to identify essential elements such as the button to explore device details and the pairing code input box. Consequently, it could not perform device pairing or identify *UMCCI* pages. In contrast, *UMCCI* Checker features accurate UI element identification and filter that ensure no necessary elements are overlooked, enabling thorough exploration.

TABLE I: Comparing with SOTA Tools

APP Name	**UMCCI Checker**		**AutoDroid**		**DroidBot**	
	T1	**T2**	**T1**	**T2**	**T1**	**T2**
Apple Home	✓	✓	×	×	—	—
Google Home	✓	✓	✓	✓	—	—
SmartThings	✓	✓	×	×	—	—
Amazon Alexa	✓	✓	×	×	—	—
Tuya Smart	✓	✓	×	×	—	—
Smart Life	✓	✓	×	×	—	—
uHome+	✓	✓	×	×	—	—
Aqara Home	✓	✓	✓	✓	—	—
WiZ V2	✓	✓	✓	✓	—	—
Nanoleaf	✓	✓	×	×	—	—
Failed	0		7		—	

Discussion. *UMCCI* Checker did not encounter false positives in reporting *UMCCI* flaws in testing the 10 IoT apps. Potential false positives may arise if *UMCCI* Checker fails to correctly identify the *UMCCI* pages, and in such a case, detecting *UMCCI* flaws on wrong *UMCCI* pages may lead to false alarms. Thanks to our search prompt template, the search and identification of *UMCCI* pages were correct across all 10 apps.

Generality of *UMCCI* flaw analysis. Our analysis and *UMCCI* Checker can be generalized to detect flaws in user-facing control interfaces across various protocols. For other protocols, end-users should reliably view/update access-control statuses based on security requirements like *P*1-*P*3. The application-layer semantic elements to analyze may differ across protocols, like fabrics in Matter. For instance, in user-facing "Bluetooth" control capabilities and interfaces (*UBCCI*), we could verify attributes of Bluetooth device and controller (called "peripheral" and "central" [24]), such as device name, profile configuration, and icon, all of which are access control-related and susceptible to forgery or modification by attackers [56].

VII. Related Work

Security analysis of the Matter protocol. Previous studies on the Matter protocol [42], [45], [50] have primarily addressed isolated incidents and case studies, and potential display errors, providing only a limited scope of the associated security risks. Our research is new and systematic, introduces the *UMCCI* flaws, analyzes the Matter development model, and based on the model systematically identifies novel, subtle design flaws of both vendors and Matter, and introduces new serious attack vectors in both inter-fabric and intra-fabric Matter systems. Additionally, we introduce a novel technique *UMCCI* Checker to automatically analyze *UMCCI* flaws from real vendors' apps and further propose effective mitigation strategies to address these vulnerabilities.

Security analysis of other IoT protocols. Recent studies have extensively examined various IoT protocols, revealing key vulnerabilities and proposing security enhancements across different platforms and systems [39], [40], [43], [57], [58], [62]. These works have identified issues ranging from authentication gaps in MQTT to inadequate policy enforcement in cloud-based systems. However, none have specifically addressed the unique challenges posed by the Matter protocol. In contrast, our work specifically focuses on the Matter protocol, identifying novel security risks and proposing targeted strategies to mitigate these vulnerabilities.

Security analysis of IoT platforms, devices and applications. The security of IoT platforms and applications/devices has been extensively explored in recent studies [19], [23], [25], [26], [28], [29], [33], [35], [35], [36], [36]–[38], [41], [46], [49], [51]–[53], [58]–[61], [61]. These studies predominantly focus on specific cloud platforms and application layer logic errors. In contrast, our research systematically examines security issues related to the Matter protocol, addressing gaps in current IoT security research.

VIII. CONCLUSION

This paper systematically investigates the security risks of *UMCCI* in the Matter-based IoT environment, which is assisted by a novel automatic tool called *UMCCI* Checker we developed. This has enabled us to understand the fundamental challenges within *UMCCI* and identify various root causes that can lead to flaws in real-world systems. These security flaws significantly compromise Matter users' control and awareness of their devices. Our research indicates that this security weakness is both widespread and fundamental, affecting both vendor and Matter design. Thus, we propose immediate fix suggestions. Our research on the security of the Matter protocol has pioneered further study in this field.

ACKNOWLEDGMENT

Luyi Xing is supported in part by NSF CNS-2145675. Haoqiang Wang, Yiwei Fang, Ze Jin, and Qixu Liu are supported in part by the Youth Innovation Promotion Association CAS, the Key Laboratory of Network Assessment Technology at Chinese Academy of Sciences and Beijing Key Laboratory of Network security and Protection Technology.

REFERENCES

[1] "User interface tests — apple developer documentation," https://developer.apple.com/documentation/xctest/user_interface_tests, accessed: 2024-07-03.

[2] "Add support for Matter in your smart home app - WWDC21 - Videos - Apple Developer," https://developer.apple.com/videos/play/wwdc2021/10298/, 2021.

[3] "MatterAddDeviceRequest — Apple Developer Documentation," https://developer.apple.com/documentation/mattersupport/matteradddevicerequest, 2021.

[4] "The Matter protocol - CSA-IOT," https://csa-iot.org/newsroom/smart-home-innovation-set-to-accelerate-with-matter/, 2022.

[5] "Matter handbook," Online, 2023, accessed: 2024-4-24. [Online]. Available: https://handbook.buildwithmatter.com/

[6] "Amazon Alexa," https://alexa.amazon.com/, 2024.

[7] "Best Google Home Devices: Smart Gadgets and Products for Your Home — Google Home," https://home.google.com/welcome/, 2024.

[8] "Common vulnerabilities and exposures (cve)," https://www.cve.org/, 2024, accessed: 2024-09-28.

[9] "Distributed Compliance Ledger," https://webui.dcl.csa-iot.org/, 2024.

[10] "Home app - Apple," https://www.apple.com/home-app/, 2024.

[11] "Matter — Apple Developer Documentation," https://developer.apple.com/documentation/matter, 2024.

[12] "Matter support in iOS 16 - Apple Home - Apple Developer," https://developer.apple.com/apple-home/matter/, 2024.

[13] "The paper website," https://sites.google.com/view/mattercontrollerflaws, 2024.

[14] "Tuya Developer - Tuya Smart - Global IoT Development Platform," https://developer.tuya.com/, 2024.

[15] "Tuya Smart - Global IoT Development Platform Service Provider," https://www.tuya.com, 2024.

[16] "Uascent Technology Company Limited," https://www.uascent-iot.com/, 2024.

[17] "WiZ: Home — Smart lighting for your daily living," https://www.wizconnected.com, 2024.

[18] C. S. Alliance, "Csa iot members," 2024. [Online]. Available: https://csa-iot.org/members/

[19] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1362–1380. [Online]. Available: https://doi.org/10.1109/SP.2019.00013

[20] Android Developers, "Androidx test ui automator," https://developer.android.com/reference/androidx/test/uiautomator/package-summary, 2024, accessed: 2024-07-03.

[21] Appium, "Appium: Mobile App Automation Made Awesome," https://appium.io/, accessed: 2024-07-03.

[22] Apple, "Use iCloud Keychain to keep information safe on your Mac, iPhone, and iPad," 2023, accessed: 2023-12-30. [Online]. Available: https://support.apple.com/en-us/102135

[23] I. Bastys, M. Balliu, and A. Sabelfeld, "If this then what?: Controlling flows in iot apps," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 1102–1119. [Online]. Available: https://doi.org/10.1145/3243734.3243841

[24] Bluetooth SIG, "Bluetooth core specification 5.4," https://www.bluetooth.com/specifications/specs/core54-html/, 2023, accessed: 2024-10-28.

[25] Z. B. Celik, P. D. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, H. S. Gunawi and B. C. Reed, Eds. USENIX Association, 2018, pp. 147–158. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/celik

[26] Z. B. Celik, G. Tan, and P. D. McDaniel, "Iotguard: Dynamic enforcement of security and safety policy in commodity iot," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/iotguard-dynamic-enforcement-of-security-and-safety-policy-in-commodity-iot/

[27] ChatGPT, "ChatGPT: Your AI Conversation Partner," https://chatgpt.com/, accessed: 2024-07-03.

[28] J. Chen, C. Zuo, W. Diao, S. Dong, Q. Zhao, M. Sun, Z. Lin, Y. Zhang, and K. Zhang, "Your iots are (not) mine: On the remote binding between iot devices and users," in *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019*. IEEE, 2019, pp. 222–233. [Online]. Available: https://doi.org/10.1109/DSN.2019.00034

[29] 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 *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM, 2020, pp. 1699–1716. [Online]. Available: https://doi.org/10.1145/3372297.3423339

[30] P. CHIP, "Chip - examples - github," 2024. [Online]. Available: https://github.com/project-chip/connectedhomeip/tree/master/examples

[31] ——, "project-chip/connectedhomeip - github," 2024. [Online]. Available: https://github.com/project-chip/connectedhomeip

[32] ——, *Working with the CHIP Tool*, 2024, accessed: 2024-10-30. [Online]. Available: https://project-chip.github.io/connectedhomeip-doc/development_controllers/chip-tool/chip_tool_guide.html

[33] J. Choi, H. Jeoung, J. Kim, Y. Ko, W. Jung, H. Kim, and J. Kim, "Detecting and identifying faulty iot devices in smart home with context extraction," in *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25-28, 2018*. IEEE Computer Society, 2018, pp. 610–621. [Online]. Available: https://doi.org/10.1109/DSN.2018.00068

[34] Connectivity Standards Alliance. (2024, Apr.) Matter 1.3 core specification. [Online]. Available: https://csa-iot.org/wp-content/uploads/2024/05/Matter-1.3-Core-Specification.pdf

[35] W. Ding and H. Hu, "On the safety of iot device physical interaction control," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 832–846. [Online]. Available: https://doi.org/10.1145/3243734.3243865

[36] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 636–654. [Online]. Available: https://doi.org/10.1109/SP.2016.44

[37] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 531–548. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/fernandes

[38] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decentralized action integrity for trigger-action iot platforms," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_01A-3_Fernandes_paper.pdf

[39] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang, "Burglars' iot paradise: Understanding and mitigating security risks of general messaging protocols on iot clouds," in *2020 IEEE Symposium on Security and Privacy (SP)*, May 2020, pp. 465–481.

[40] Y. Jia, B. Yuan, L. Xing, D. Zhao, Y. Zhang, X. Wang, Y. Liu, K. Zheng, P. Crnjak, Y. Zhang, D. Zou, and H. Jin, "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*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1289–1305. [Online]. Available: https://doi.org/10.1145/3460120.3484592

[41] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. Unviersity, "Contexlot: Towards providing contextual integrity to appified iot platforms." in *ndss*, vol. 2, no. 2. San Diego, 2017, pp. 2–2.

[42] Z. Jin, Y. Fang, H. Wang, K. Chen, Q. Liu, and L. Xing, "Wip: Delegation related privacy issues in matter," in *Workshop on Security and Privacy in Standardized IoT (SDIoTSec) 2024 Program*, 2024.

[43] Z. Jin, L. Xing, Y. Fang, Y. Jia, B. Yuan, and Q. Liu, "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*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1647–1661. [Online]. Available: https://doi.org/10.1145/3548606.3560680

[44] Y. Li, Z. Yang, Y. Guo, and X. Chen, "Droidbot: a lightweight ui-guided test input generator for android," in *Proceedings of the 39th International Conference on Software Engineering Companion*, ser. ICSE-C '17. IEEE Press, 2017, p. 23–26. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.8

[45] S. Liao, J. Yan, and L. Cheng, "Wip: Hidden hub eavesdropping attack in matter-enabled smart home systems," in *Workshop on Security and Privacy in Standardized IoT (SDIoTSec) 2024 Program*, 2024.

[46] Y. Nan, X. Wang, L. Xing, X. Liao, R. Wu, J. Wu, Y. Zhang, and X. Wang, "Are you spying on me? large-scale analysis on iot data exposure through companion apps," in *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, J. A. Calandrino and C. Troncoso, Eds. USENIX Association, 2023, pp. 6665–6682. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/nan

[47] Project Connected Home over IP (CHIP), "Matter.js: A JavaScript implementation of the Matter protocol," 2024, accessed: 2024-10-24. [Online]. Available: https://github.com/project-chip/matter.js/

[48] W. Stallings and L. Brown, *Computer security: principles and practice*. Pearson, 2015.

[49] Y. Tian, N. Zhang, Y. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "Smartauth: User-centered authorization for the internet of things," in *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, E. Kirda and T. Ristenpart, Eds. USENIX Association, 2017, pp. 361–378. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tian

[50] H. Wang, Y. Liu, Y. Fang, Z. Jin, Q. Liu, and L. Xing, "Wip: Security vulnerabilities and attack scenarios in smart home with matter," in *Workshop on Security and Privacy in Standardized IoT (SDIoTSec) 2024 Program*, 2024.

[51] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, and C. A. Gunter, "Charting the attack surface of trigger-action iot platforms," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM, 2019, pp. 1439–1453. [Online]. Available: https://doi.org/10.1145/3319535.3345662

[52] Q. Wang, W. U. Hassan, A. Bates, and C. A. Gunter, "Fear and logging in the internet of things," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_01A-2_Wang_paper.pdf

[53] X. Wang, Y. Sun, S. Nanda, and X. Wang, "Looking from the mirror: Evaluating iot device security through mobile companion apps," in *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, N. Heninger and P. Traynor, Eds. USENIX Association, 2019, pp. 1151–1167. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/wang-xueqiang

[54] H. Wen, Y. Li, G. Liu, S. Zhao, T. Yu, T. J.-J. Li, S. Jiang, Y. Liu, Y. Zhang, and Y. Liu, "Autodroid: Llm-powered task automation in android," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ser. ACM MobiCom '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 543–557. [Online]. Available: https://doi.org/10.1145/3636534.3649379

[55] WiZ Connected, "How do i connect wiz lights to alexa?" n.d., accessed: 2024-10-30. [Online]. Available: https://wizconnected.helpshift.com/hc/en/7-wiz-v2/faq/540-alexa/

[56] F. Xu, W. Diao, Z. Li, J. Chen, and K. Zhang, "Badbluetooth: Breaking android security mechanisms via malicious bluetooth peripherals." [Online]. Available: https://staff.ie.cuhk.edu.hk/~khzhang/my-papers/2019-ndss-badbluetooth.pdf

[57] B. Yuan, Z. Song, Y. Jia, Z. Lu, D. Zou, H. Jin, and L. Xing, "Mqttactic: Security analysis and verification for logic flaws in mqtt implementations," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 13–13. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00013

[58] B. Yuan, Y. Jia, L. Xing, D. Zhao, X. Wang, and Y. Zhang, "Shattered chain of trust: Understanding security risks in Cross-Cloud IoT access delegation," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1183–1200. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/yuan

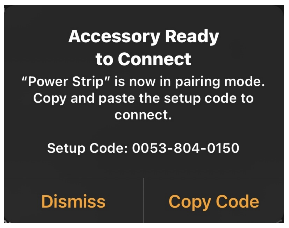
[59] B. Yuan, Y. Wu, M. Yang, L. Xing, X. Wang, D. Zou, and H. Jin, "Smartpatch: Verifying the authenticity of the trigger-event in the iot platform," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 2, pp. 1656–1674, 2023. [Online]. Available: https://doi.org/10.1109/TDSC.2022.3162312

[60] 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 *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1381–1396. [Online]. Available: https://doi.org/10.1109/SP.2019.00016

[61] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang, "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 2019, Santa Clara, CA, USA, August 14-16, 2019*, N. Heninger and P. Traynor, Eds. USENIX Association, 2019, pp. 1133–1150. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/zhou

[62] X. Zhou, J. Guan, L. Xing, and Z. Qian, "Perils and mitigation of security risks of cooperation in mobile-as-a-gateway iot," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 3285–3299. [Online]. Available: https://doi.org/10.1145/3548606.3560590

APPENDIX


Accessory Ready to Connect
"Power Strip" is now in pairing mode. Copy and paste the setup code to connect.

Setup Code: 0053-804-0150

Fig. 10: Matter Sharing UI in Apple Home

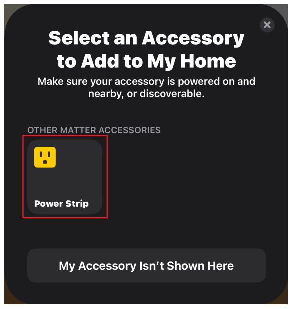

Select an Accessory to Add to My Home
Make sure your accessory is powered on and nearby, or discoverable.

Fig. 11: Tapping the icon pairs devices from the Keychain fabric with Apple Home.

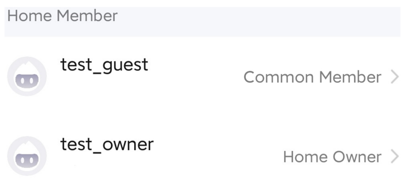
```
{
"gid": 166897639 ,
"elementid": 166897639 ,
"pubkey": ###
}
```

(a) Request

```
{
"fabricid": 4540950 ,
"icac": ### ,
"nodeid": 65537 ,
"rcac": ###
}
```

(b) Response

Fig. 12: Request and Response JSON Format of Tuya API thing.m.matter.icac.issue


Home Member

test_guest	Common Member
test_owner	Home Owner

Fig. 13: Tuya app displays the owner as still having Home Owner status after their ACL privileges have been downgraded.

A. *Prompts of UMCCI Checker*

The Crafted LLM Pairing Prompt

### Role: System
You are a smartphone assistant to help users complete tasks by interacting with mobile apps related to IoT control based on the Matter protocol (just like Tuya Smart, Google Home, Amazon Alexa and so on). Your task is to cooperate in the depth-first exploration of the app page elements, and pair a matter device with setup code XXXX-XXX-XXXX. The order of exploration of the elements on the page is sorted according to importance, with important elements at the front.
Given a list containing all elements of the current UI page, your job is to reorder the list based on your prediction of the importance of each element in the list. At the same time, you should pay attention that when you analyze that the element may be a return, back page, exit, or similar operation, put such operations at the end to reduce invalid exploration such as exiting the page too early. Likewise, if you find specific pairing device icons related elements, put them in the front position for exploration.
Your answer must always be one element per line, without numbering (just like the list of elements given to you each time). Your answer may not contain any explanatory text. The number of elements you output must match the number of elements in the original list given to you. You cannot omit elements or modify the content of elements.
### Role: User
The list of elements is as follows:
<xxx><xxx>...
If you think the task has been completed, please output '0'.

The Crafted LLM Searching Prompt

### Role: System
You are a smartphone assistant to help users complete tasks by interacting with mobile apps related to IoT control based on the Matter protocol (just like Tuya Smart, Google Home, Amazon Alexa and so on). Your task is to cooperate in the depth-first exploration of the app page elements. The order of exploration of the elements on the page is sorted according to importance, with important elements at the front. Additionally, you need to determine whether the current page is a UMCCI page. Typically, the page title resembles "Connected Services" or similar synonyms, and it displays a list of all Matter controllers with parameters such as vendor name or ID and Label value.
Given a list containing all elements and the screenshot of the current UI page, your job is to reorder the list based on your prediction of the importance of each element in the list. After completing the list output, you need to output the result of whether the page is a UMCCI page on the last line. If it is,

TABLE II: Design Flaws in User-Facing Matter-Controllers of Eight Vendors

UMCCI Flaws		Apple	Google	Amazon Alexa	Samsung SmartThings	Tuya	Aqara	Uascent	Signify WiZ
Vendor Design	Flaw Type 1	✗	✓	N/A	✗	✓	✓	✓	N/A
	Flaw Type 2	✗	✗	N/A	✗	✗	✗	✗	N/A
	Flaw Type 3	✓	✓	✗	✓	✓	✓	✓	✗
Matter Design	Flaw Type 4	✗	✓	✓	✓	✗	✓	✓	N/A
	Flaw Type 5	✓	✓	✓	✓	✗	✓	✗	N/A
	Flaw Type 6	✓	✓	✓	✓	✓	✓	✓	✗

✗ denotes that we successfully exploited the weakness in the Matter controller of the specified vendor. ✓ denotes that we were unable to exploit the weakness in the Matter controller of the specified vendor. N/A denotes that the vendor did not implement the related Matter feature.

TABLE III: The Matter Devices of Our Study

Vendor	Device Model	IoT App Name
Apple	HomePod mini	Home
Google	Nest mini	Google Home
Samsung SmartThings	Station EP-P9501	SmartThings
Amazon Alexa	Echo Dot 3	Amazon Alexa
Tuya	Gateway THP10-Z-X	Tuya Smart/Smart Life
Uascent	LED Light	uHome+
Aqara	M2 Hub 2002	Aqara Home
Signify WiZ	A19 Light	WiZ V2
Zemismart	LED E27	N/A
Nanoleaf	A60 Smart Bulb	Nanoleaf
Xenon	SM-PW703 Smart Plug	N/A

TABLE IV: Examples of CHIP SDK APIs

API Name	Arguments	Description
commissioner-name	[Fabric_ID]	Set the controller's Fabric ID when interacting with devices.
commissioner-nodeid	[Node_ID]	Set the controller's Node ID when interacting with devices.
commissioner-vendorid	[Vendor_ID]	Set the controller's Vendor ID.
pairing code	[Node_ID] [Pairing_Code]	Pair with a new device through pairing code.
operationalcredentials update-fabric-label	[Label_Value] [Node_ID] [Endpoint_ID]	Set the controller's fabric Label.
operationalcredentials read fabrics	[Node_ID] [Endpoint_ID]	Queries all Fabrics of the device.
pairing open-commissioning-window	[Node_ID] [Option] [Iteration] [Discriminator]	Open the device's commissioning window and enter the device pairing process.
accesscontrol read acl	[Node_ID] [Endpoint_ID]	Read ACL in the current fabric.
accesscontrol write acl	[ACL_Data] [Node_ID] [Endpoint_ID]	Write ACL to the current fabric.
binding write binding	[Binding_Data] [Node_ID] [Endpoint_ID]	Describe a relationship between the device that contains the binding cluster and the end device.

output 'yes'; if it is not, output 'no'. At the same time, you should pay attention that when you analyze that the element may be a return, back page, exit, or similar operation, put such operations at the end to reduce invalid exploration such as exiting the page too early. Likewise, if you find specific device names or device icons related elements, put them in the front position for exploration.
Your answer must always be one element per line, without numbering (just like the list of elements given to you each time). Your answer may not contain any explanatory text. The number of elements you output must match the number of elements in the original list given to you. You cannot omit elements or modify the content of elements.
### Role: User
The list of elements is as follows:
<xxx><xxx>...
If you think the task has been completed, please output '0'.

The Crafted LLM Checking Prompt

### Role: System
You are an Interface Analyst for IoT apps based on the Matter protocol, your responsibilities include:
1. Analyze the app interface screenshots I provide to determine whether they contain specific information I have designated.
2. Based on the presence of this information, evaluate the vulnerabilities or security aspects of the app related to the displayed interface.
3. Offer specific recommendations for remediation based on the type of vulnerabilities identified.
You need to verify the presence of the following information in the screenshot:
1. Check if there is any mention of "current app" or synonymous phrases.
2. Determine whether the number of Matter connection services shown is two.
3. Assess whether the Matter connection services include the special Label value.
4. Verify the availability of ACL query functionality.
5. Check for the presence of the phrase "Node ID".
6. Determine if "Vendor ID" or the vendor's name is displayed.
Your remediation suggestions should be as follows:
- If 1 is absent, suggest "Mark the current Matter controller clearly."
- If 2 is absent, suggest "Display all entries of Matter devices fully; none should be hidden."
- If 3 is absent, suggest "Display complete Fabric Label values."
- If 4 is absent, suggest "Include functionality for ACL information query."
- If 5 is absent, suggest "Display complete Node ID information within the Fabric."
- If 6 is absent, suggest "Mark the Matter controller's Vendor ID or the vendor's name."
The output format you should follow is:
`{"flaws":"","suggestions":""}`
You cannot output any other text except this format!
Example and explanation provided:

```
{
 "flaws": "110001",
 "suggestions": "Display complete
 Fabric Label values. Include func-
 tionality for ACL information query.
 Display complete Node ID informa-
 tion within the Fabric."
}
```

In this output, 'flaws' means that points 1, 2, and 6 were confirmed as present ('1'), whereas points 3, 4, and 5 were absent ('0'), resulting in "110001". The 'suggestions' are the

respective remediation advice for points 3, 4, and 5 linked together, as explained.
If ‘flaws’ results in “111111”, then leave ‘suggestions’ blank.
### Role: User
The attached picture is the picture I provided to you, please analyze it.
You can only output the JSON string `{"flaws":"", "suggestions":""}`. Do not output any other text. Do not output your analysis process. Do not output any textual explanation.

B. Mitigation Recommendations of UMCCI Checker

MITIGATION RECOMMENDATIONS

Vendor design. The main reason for vendors encountering user interface flaws is due to design issues when integrating the Matter protocol into their existing app environment. In response to the flaws identified in § IV, we propose the following defense recommendations:

• When vendors design unique IoT features based on the Matter protocol, they must consider whether these features conflict with existing Matter characteristics. It is crucial to avoid flaws in the design of new functionalities that could lead to missing or incorrect information in the user interface. [Flaw Type 1]

• The Matter protocol is inherently local. When integrating IoT cloud service frameworks, vendors must pay close attention to the rigour of user interface logic and information synchronization between Matter and other protocol. This ensures that device owners can accurately and timely understand the status of devices at home through the user interface, preventing malicious actions from being executed invisibly. [Flaw Type 2]

• Vendors using the Matter protocol should fully implement critical functionalities, especially those related to device pairing and permission management in the user interface. The design should also consider the users' habits and the practicality of interactions with IoT devices. [Flaw Type 3]

Matter design. The main problem with the Matter protocol is the lack of sufficient guidance for vendors on user interface design, along with certain features in the protocol that are not optimally designed. In response to the flaws identified in § V, we propose the following defense recommendations:

• Given that the Matter protocol currently allows for arbitrary customization of controller and fabric data, such as Vendor ID and Label, solely through API access, there should be mechanisms introduced to verify the legitimacy of these data. Similar to the existing Distributed Compliance Ledger (DCL) that monitors Matter devices, certification or online checks could be employed to promptly detect forged information. [Flaw Type 4]

• The Matter protocol should implement a system for recording and managing node information. This would not only facilitate the management of paired device nodes for vendors and users but also prevent the invisibility of malicious devices, thus enhancing the security of the Matter protocol. [Flaw Type 5]

• The Matter protocol needs to strengthen its guidance and constraints for vendors. Developer documentation should clearly indicate which information is essential to be displayed to users and how certain functionalities differ from existing IoT protocol implementations. This clarity will help reduce the security risks associated with the highly autonomous development processes by vendors. [Flaw Type 6]

Additionally, the CSA should enhance the information in the following sections of *Matter Specification* version 1.3 [34] document to provide more rigorous guidance for vendor development:

Essential Device Information Visibility [Flaw Type 1, 2, 4, 6]

• VendorID, FabricID, NodeID, Label in Section 11.18.4.5

The data structures FabricDescriptorStruct Type encodes a Fabric Reference for a fabric within which a given Node is currently commissioned. Two of these Fields, Vendor ID and Label, must be represented in the app UI of the vendor's design. A NOTE should be made here in the document.

• CurrentFabricIndex in Section 11.18.5

This data structure is used to output the current fabric index value. When a vendor's Matter controller queries the connected services of a Matter device, it should display and mark which one is the current controller (fabric) in the list, to prevent attackers from creating a malicious fabric and confusing the device's owner.

Essential Controller Functionality [Flaw Type 3, 6]

• FabricDescriptorStruct in Section 11.18.4.5

• RemoveFabric in Section 11.18.6.12

When implementing the Matter controller functionality in their clients, vendors must ensure it is complete. Essential features must include the capability to query the fabric of the Matter devices (i.e., the services to which the devices are connected). In addition to correctly displaying information about each fabric, it is also necessary to have the ability to remove a fabric. This section should include a NOTE to highlight this requirement.

Error Description Correction [Flaw Type 6]

Reserved Vendor ID Range Error in Section 11.18.6.8 and Section 2.5.2 Section 11.18.6.8 mentions that the value of AdminVendorID should not be one of the reserved Vendor IDs listed in Table 1 of Section 2.5.2. However, Section 2.5.2 states that all Vendor IDs in Table 1 are reserved, which implies that no Vendor IDs are available for use. This creates a contradiction in the descriptions between these sections.