Demystifying the Security Implications in IoT Device Rental Services

Yi He[†*], Yunchao Guan[†*], Ruoyu Lun[♮], Shangru Song[†], Zhihao Guo[†], Jianwei Zhuge[†‡✉], Jianjun Chen[†‡], Qiang Wei[♮], Zehui Wu[♮], Miao Yu[†], Hetian Shi[†], Qi Li[†‡✉]

[†]Tsinghua University, ✉zhugejw@tsinghua.edu.cn, ‡Zhongguancun Laboratory, ✉qli01@tsinghua.edu.cn
[♮]China National Digital Switching System Engineering and Technological Research Center

Abstract

Nowadays, unattended device rental services with cellular IoT controllers, such as e-scooters and EV chargers, are widely deployed in public areas around the world, offering convenient access to users via mobile apps. While differing from traditional smart homes in functionality and implementation, the security of these devices remains largely unexplored. In this work, we conduct a systematic study to uncover security implications in IoT device rental services. By investigating 17 physical devices and 92 IoT apps, we identify multiple design and implementation flaws across a wide range of products, which can lead to severe security consequences, such as forcing all devices offline, remotely controlling all devices, or hijacking all users' accounts of the vendors. The root cause is that rentable IoT devices adopt weak resource identifiers (IDs), and attackers can infer these IDs at scale and exploit access control flaws to manipulate these resources. For instance, rentable IoT products allow authenticated users to find and use any device from the rentable IoT apps via a device serial number, which can be easily inferred by attackers and combined with other vulnerabilities to exploit remote devices on a large scale. To identify these risks, we propose a tool, called IDScope, to automatically detect the weak IDs in apps and assess if these IDs can be abused to scale the exploitation scope of existing access control vulnerabilities. Finally, we identify 57 vulnerabilities in 28 products which can lead to various large-scale exploitation in 24 products and affect millions of users and devices by exploiting three types of weak IDs. The vendors have confirmed our findings and most issues have been mitigated with our assistance.

1 Introduction

The sharing economy [10] is a new trend in cities, providing unattended IoT device rental services to consumers.By integrating cellular Internet of Things (IoT) controllers into useful tools such as e-scooters, chargers, and washing machines, these devices can be rented to users and billed based on their usage duration in an unattended manner. Benefiting from IoT controllers, these devices can be controlled remotely, enabling operators to efficiently manage and monitor large numbers of devices over cellular networks. Users can conveniently rent these devices on a self-service basis via the apps to enhance their travel, entertainment and life experiences. With an increasing number of rentable mobility and charging devices deployed in cities, misuse of these devices can result in serious consequences, such as property damage, and jeopardizing public safety. However, the security implications of these rentable IoT devices remain under-explored [59].

Previous work has demonstrated that smart home devices face significant security challenges in multi-user scenarios [53, 55, 69]. In contrast to traditional smart home devices, where attackers can only affect a limited number of devices [54, 69, 70] belonging to individual owners, IoT device rental scenarios involve a single vendor managing a large number of devices accessible to all users, rendering them more vulnerable to large-scale exploitation. Paid users can rent devices by using the devices' QR codes or serial numbers (SNs). This enables attackers to exploit remote devices via their SNs. For instance, adversaries who can bypass the payment can remotely abuse devices by obtaining their SN. Unfortunately, this security risk is not fully understood by device vendors, as many devices employ short serial numbers to simplify the entry process for users, inadvertently allowing attackers to infer these SNs, thereby facilitating widespread remote attacks across numerous devices. As a result, vulnerabilities in these rentable devices can easily lead to large-scale attacks affecting large numbers of devices and users.

Compared to existing smart home devices, rentable IoT devices require different security analysis and methodologies to address their unique security risks due to their different functionality and implementation. Unlike Wi-Fi-connected smart homes, rentable IoT devices utilize cellular networks, greatly impeding the interception or analysis of device-to-server com-

*The first two authors contributed equally to this paper.

munication. For smart home devices, insider attackers can easily eavesdrop on the traffic [13] and send spoofing requests [56, 67, 69] in smart homes, while targeting rentable IoT devices with 4G/5G cellular networks requires deploying fake base stations and exploiting vulnerabilities in the 4G/5G protocols [32, 65]. The adaptation of cellular networks highlights the need for a dedicated study of new attack strategies on emerging rentable IoT devices. Rather than adopt trigger-action platforms [16, 19, 20] or mobile-as-a-gateway [70] to control IoT devices, rentable IoT devices are directly controlled by their backend services. Existing IoT researches mainly focus on analyzing traditional IoT platforms (such as SmartThings [19, 20] or AWS IoT [35, 67]), which typically use HTTPS or MQTT protocols. Rentable devices usually implement custom backends and protocols that require new analysis methods.

To understand the security of these devices, we conduct a systematic study on real-world IoT device rental services. There are three challenges in efficiently identifying vulnerabilities in both the IoT controller and mobile apps of rentable devices. First, some devices (e.g., chargers) do not use batteries but work with high-voltage household or industrial alternating current (AC). It is dangerous to test or debug these devices to extract their firmware. Second, it is hard to analyze the communication of these devices as they adopt cellular networks and customized protocols. Third, all of these products are closed-source and it is difficult to identify threats such as large-scale off-path attacks that exploit vulnerabilities in multiple components. To address these challenges, we propose a general workflow for studying the devices and apps of these products. First, we partially boot the necessary MCUs of the devices by using jumper wires to provide extra power supply under a safe voltage. Then, we propose a log-based approach to efficiently reverse the protocols of these devices, which capture cellular network traffic in plain text from the cellular processor (CP) log. Finally, we perform black-box API tests to identify vulnerabilities on both the devices and the apps and propose a tool IDScope to effectively assess if these vulnerabilities can large-scale exploit users and devices.

We investigate 17 physical devices and 92 apps of IoT device rental services by reversing the protocols of these devices and apps, re-implementing their protocols, and testing their APIs by impersonating these devices or apps to communicate with their servers. We identify 23 vulnerabilities in 14 devices and 34 vulnerabilities in 23 apps. Meanwhile, we identify multiple products use weak IDs for both their devices and users, which can lead to large-scale remote exploitation via ID enumeration attacks. To assess this threat, we propose a dynamic analysis tool, called IDScope, which can automatically identify the weak IDs, find the appropriate APIs to verify the enumerated IDs, and assess the affected scope (e.g., numbers of devices and users) of existing vulnerabilities. We find that attackers can infer all device serial numbers of 60.9% (56/92) products. 84% (48/57) of the vulnerabilities can lead to large-scale exploitation of 24 rentable IoT products, resulting in large-scale privacy leakage, account hijacking, free device use, or device manipulation. Our findings have been assigned with 18 CVE/CNNVD/NVDB numbers[1] and can affect several popular products (e.g., Teld, StartCharge, YuehuoCX) with more than millions of users and 100K devices.

New Insights. Our study indicates that rentable IoT devices' QR-code (containing the device serial number) can also become an attack surface. This is because (1) paid users can choose any device via the device serial number. However, many rentable IoT devices only verify if a user is paid but do not check which device he/she uses. This may allow unauthorized users to remotely activate or deactivate other users' devices using their serial numbers. (2) The device serial numbers of many products are too short and have obvious patterns (e.g., common prefixes). Therefore, attackers can easily infer the serial numbers of new devices and verify these numbers using the apps' APIs. All they need is a QR code to obtain an initial device serial number. They can then use the app APIs to enumerate more serial numbers. By further exploiting other vulnerabilities (e.g., payment bypass), attackers can accomplish large-scale remote attacks on a product via the apps without touching any real device.

Contributions. Our contributions are as follows:

- We conduct a systematic study to uncover the security implications in rentable IoT devices. Moreover, we propose a general workflow to analyze all kinds of rentable IoT devices. Our approach successfully identifies 57 vulnerabilities affecting 28 real products.
- We find that most rentable IoT devices are severely vulnerable to ID enumeration attacks, which can turn ordinary access control flaws into high-risk attacks to remotely exploit devices at scale.
- We propose a tool dubbed IDScope to automatically detect ID enumeration vulnerabilities on rentable IoT products and show that 84% of the existing vulnerabilities can be exploited to attack all devices or users of several vendors.
- We propose a mitigation solution to prevent ID enumeration attacks and help vendors mitigate these vulnerabilities.

Ethics Consideration. We investigate the security of real-world products crucial to users' daily lives. To minimize impact, testing is limited to our private devices[2] or app accounts, ensuring that both attacker and victim scenarios remain within our private environments, avoiding disruption to vendors or users. To identify large-scale exploitation, we need to confirm if these products' IDs are enumerable. IDScope eliminates the need for brute-forcing all IDs, instead efficiently inferring the valid ID range by sending only a few requests at a very low rate, similar to web crawlers. The IDs we gather do not include private information; for instance, device IDs are visible on devices, and user IDs are simply integers or strings.

[1] We track the vulnerabilities at https://vehicle-security.github.io.

[2] Some devices are also sold to individuals (see § 2.1).

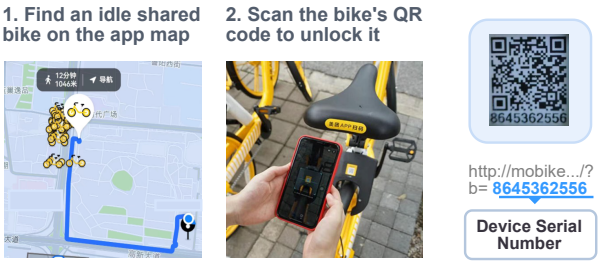


Figure 1: Users can access any device via the app.

During the research, we ensure that all of our approaches comply with the vendor's bug bounty plan [1], do not cause DoS attacks, and do not compromise user privacy.

2 Background

2.1 IoT Device Rental Service

The rise of cellular IoT technologies has popularized IoT device rental services in urban areas, allowing devices to function and be placed anywhere without relying on Wi-Fi or wired connections. As shown in Appendix A, various rental devices operate unmanned in public areas of cities and are managed unattended by cellular IoT controllers.

Cellular IoT Controllers for Rentable Device. The rentable devices are equipped with a cellular IoT controller to receive control commands from the server for manipulating the devices. The cellular IoT controller usually has the following components:

- **4G/5G MCU.** This MCU is dedicated to communicating with the cellular networks. It can also be used to process simple tasks such as data collection or send commands via Usart to control (e.g., turn on/off) physical devices (e.g., laundry machines). Thus, these devices' IoT controllers only contain the cellular MCU.
- **Main MCU (optional).** Some devices need to process complex tasks, such as managing the charging process, and adopt a dedicated MCU for real-time controlling the peripherals (e.g., DAC). We regard this task controlling MCU as the main MCU, as it usually contains the entire control logic of the devices. In devices with a main MCU, the cellular MCU is used in pass-through mode, only forwarding network packets to the server via the cellular network.

Devices Rental. All rentable devices are available to every user. Taking the popular shared bicycle as an example (see Figure 1), users can find idle bicycles on the map of the companion app and they can choose any of these bicycles (step 1). To rent a bicycle, they first need to make an order on the app by scanning the bicycle's QR code (step 2). The app then resolves the QR code's content (e.g., a URL or an ID) and sends requests to the server to remotely activate this device. After users complete the trip, they can lock the bicycle and immobilize the wheels via the apps. The cost is automatically charged on the app.

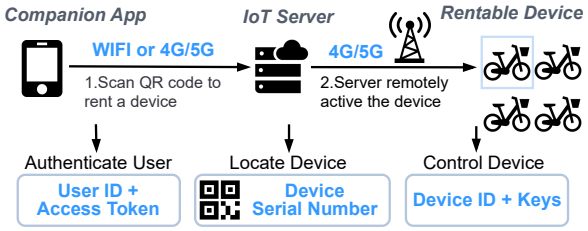


Figure 2: The workflow of device rental service.

Device Management. Mobility devices, such as e-scooters and e-bikes, are predominantly managed by large vendors who operate thousands of these units across various regions. These vendors are responsible for periodic inspections, repairs, and the strategic relocation of devices to meet the demand influenced by population density and usage patterns. In contrast, stationary devices like EV chargers and laundry machines necessitate fixed installations and a continuous power supply. These devices are purchased by individual operators, who own a few devices and strategically position them at their stores, parking facilities, or other suitable locations. This setup allows operators to rent out the devices to customers, neighbors, or others in need, generating profit while leveraging the vendor's backend systems for control and billing.

2.2 ID Enumeration Attacks

ID enumeration attacks [12] involves attackers discovering valid resource identifiers (IDs) on web or app APIs to gain unauthorized access to restricted resources by manipulating or guessing these IDs in URLs or request parameters. We find that rentable IoT devices are particularly vulnerable to ID enumeration attacks as large numbers of users and devices are managed by the same backend. As depicted in Figure 2, when the user scans a QR code to initiate a device rental request, the device's serial number is transmitted to the server. The server then authenticates the user's credentials. Subsequently, it retrieves the corresponding device information based on the provided *device serial number*. Once the target device is identified, the server transmits control commands to it via the cellular network connection. This process involves authenticating multiple resource identifiers (IDs):

- **User ID** is used by the apps to identify various users. Many rentable products automatically assign an increasing number as the user ID for the new user during registration.
- **Device Serial Number** is used by the apps to distinguish devices, which is usually very short (6-9 digits) as users sometimes need to manually enter it into the app when the QR code is abraded by malicious users [3].
- **Device ID** is stored in the firmware and used for device-server authentication. Some devices may reuse the device serial number as the device ID for simplicity.

[3]The shared bicycle market in China used to be highly competitive, resulting in frequent cases of bicycle vandalism.

We find that in multiple rentable products, the enumeration of three types of IDs can lead to severe consequences. First, user ID enumeration can lead to large-scale data breaches or account hijacking, when attackers can compromise access tokens. Second, harvesting device serial numbers can facilitate the large-scale exploitation of remote devices. For instance, if attackers can circumvent payment systems or exploit privilege escalation vulnerabilities [45], they can manipulate all devices identified by the enumerated serial numbers. Furthermore, we have observed that numerous devices share common authentication keys or lack proper device authentication, allowing server connections solely based on device IDs. These vulnerabilities make these devices susceptible to large-scale impersonation attacks using enumerated device IDs. We provide real-world examples of such attacks and illustrate how attackers can remotely exploit users or devices from various vendors in sections § 6.3 and § 6.4.

3 Overview

3.1 Threat Model

We consider rentable IoT devices and their smartphone apps as potential attack targets and focus on remote exploits that do not require attackers to have physical access to the victim devices.The attacker's goal is to use devices for free or do damage to the device (e.g., DoS) or other users (e.g., information stealing, account hijacking). To achieve this goal, they may try to exploit the communication between the devices and the server, or communication between the app and the server to affect the devices or other users. Meanwhile, attackers may try to reverse-engineer and dynamic/static analyze both the device firmware and the apps. After attackers grasp the devices' and apps' credentials and protocols, they may off-path exploit the server APIs by impersonating the devices or apps to send spoofed requests to the server.

Our only assumption is that attackers can analyze both devices and apps. This is reasonable because rentable IoT devices are usually deployed unattended in public spaces, making it easy for attackers to illegally acquire one. Note that they only need one device[4] to explore the firmware and extract the secrets. IoT apps are also publicly available on app markets or can be accessed in WeChat. Attackers can readily manipulate these apps from malicious mobile devices, such as rooted Android phones, using their own accounts.

3.2 Methodology

As shown in Figure 3, we developed a semi-automated workflow to analyze rentable IoT devices. We first combine reverse engineering and black-box API testing to find vulnerabilities in apps or devices. Then, we use IDScope to assess if these

[4]In our study, we use two devices, one acting as the attacker and the other as the victim, to avoid ethical issues.

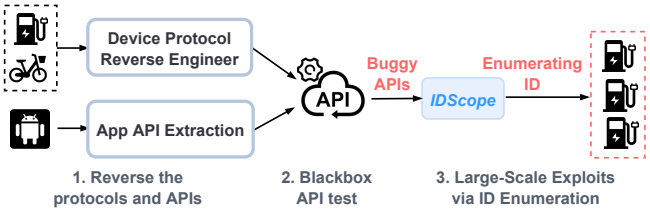


Figure 3: The workflow of identifying large-scale exploitation vulnerabilities in devices and apps of the rentable products.

vulnerabilities can be scaled to remotely exploit more devices at a large scale. The detailed steps are as follows:

S1: Reverse Engineering Protocols and APIs. This step aims to reverse-engineer the APIs and extract the credentials (e.g., crypto keys and access tokens) for further testing these APIs. The companion apps usually use HTTPS and can be easily analyzed via MitM. However, most rentable IoT devices adopt custom binary protocols to communicate with the IoT server. Besides capturing their network traces, we also need to further analyze their firmware to extract the cryptographic algorithms and keys (illustrated in § 4.1).

S2: Black-Box API Testing. In this step, we focus on identifying vulnerabilities that can affect the current devices or app users. We use a semi-automated approach to identify insecure server APIs when they communicate with apps or devices. For the physical devices, we manually check their server connection protocols to verify if the devices are correctly authenticated by the servers. For instance, some devices do not encrypt the messages or do not use unique keys to authenticate different devices. For the companion Apps, we automatically replay the existing requests and substitute the identifier (e.g., `uid`, `Device Serial Number`) fields to check if the server can correctly authenticate and authorize [72] these APIs. By checking the return code, we can identify the vulnerable APIs and then manually check their impacts. The analysis results are illustrated in § 4.2 and § 5.

S3: Large-Scale Exploits via ID Enumeration. Many simple vulnerabilities can be scaled to attack many other devices remotely when attackers can enumerate devices' or Users' IDs. This is because some devices have weak authentications (discussed in § 4) such as sharing the same cryptographic keys, which allows the attackers to large-scale impersonate any device if they know other devices' IDs. The same situation occurs in the apps, as some APIs do not strictly validate the users, and attackers can exploit these vulnerable APIs to misuse devices. We propose IDScope to assess large-scale exploitation of the devices or the apps in § 6.

3.3 Common Weakness in Rentable Products

We systematically study the common vulnerabilities in the design and implementation of IoT device rental services by investigating 17 devices and 92 apps. More than 50 vulnerabilities are identified in the devices or apps, leading to large-scale

Table 1: Common weakness in rental services' IoT devices or apps.

#Products	#	Common Weakness	Security Impacts	#Vul Devices
Device (17, see Table 2)	D1	The log port of cellular chips is unprotected	Intra/Inter-Chip network traffic capture	17 (100%)
	D2	No protection for MCU debug interfaces	Debug and extract the firmware	9 (52.9%)
	D3	Unstripped firmware with log strings	Facilitate protocol analysis	14 (82.4%)
	D4	Insecure authentication implementation	Device impersonate (see Table 3)	8 (47.1%)
	D5	Multiple devices share the same secrets	Device impersonate (see Table 3)	4 (23.5%)
	D6	Device ID is enumerable	Large-Scale device impersonate (see Table 6) *	11 (64.7%)
App (92, see Table 8)	A1	Vulnerable payment implementation	Free device usage (see Table 7)	5 (5.4%)
	A2	Improper permission checks of app APIs	Privacy leakage, Free device usage (see Table 4)	15 (16.3%)
	A3	User ID is enumerable	Large-Scale exploits other users (see Table 7) *	14 (15.2%)
	A4	Device Serial Number (SN) is enumerable	Large-Scale exploits other devices (see Table 7) *	56 (60.9%)

* Without other vulnerabilities, ID enumeration is only a security risk. For large-scale attacks, D6 relies on D4/D5 and A3/A4 relies on A1/A2.

privacy leakage, device manipulation, and free device usages, which are detailed in § 4, § 5, and § 6. We summarize the underlying weaknesses that lead to these vulnerabilities in Table 1.

Weakness in Device Hardware and Firmware (D1-D3). Rentable IoT devices lack detection for hardware tampering via software (**D1**), fail to secure the chips' debug modes (**D2**), and retain log printing code in their release version (**D3**). Although these vulnerabilities cannot lead to direct exploitation, attackers can employ them to dynamically analyze these devices to grasp their communication with the servers for further identifying remote exploitation vulnerabilities.

Weakness in Device-Server Interactions (D4-D6). We identify several devices that use insecure authentication (**D4**) or encryption (**D5**). Devices with these vulnerabilities are only distinguished and authenticated by the server using their unique device identifiers [64, 69]. If the attackers can gain other devices' identifiers, they can impersonate all these devices. We identify several devices' identifiers that can be enumerated by attackers (**D6**), allowing attackers to remotely exploit as many devices as they can enumerate.

Weakness in App-Server Interactions (A1-A4). We find that several companion apps have logic bugs in their payments allowing users to rent the devices without paying (**A1**). Some devices have APIs with improper permission checks, allowing unauthorized access (**A2**). These vulnerabilities can be scaled to all devices when attackers can enumerate (**A3**) all devices' serial numbers (i.e., QR codes). Some products' user IDs can also be enumerated and expand the scope of horizontal privilege escalation to exploit all users (**A4**).

4 Analyzing Vulnerabilities of Devices

4.1 Workflow for Studying Rentable Devices

To investigate device security, it is essential to have physical access to real rentable devices and disassemble their IoT controllers to perform tasks such as connecting jumper wires [24] to chips for network and firmware analysis. However, most rentable devices are owned by vendors or operators (e.g., all Uber e-scooters belong to Uber), making it challenging to acquire devices for research purposes. Fortunately, some devices (e.g., Teld EV-Charger) are available for individual purchase, allowing users to participate in the vendor's IoT rental service and rent out devices for profit. Ideally, our approach requires only two devices for each product, one acting as the attacker and the other as the victim.We buy 16 rentable IoT devices (shown in Table 2) and cooperate with a vendor to gain an extra device (i.e., *Meituan* bicycle).

We employ protocol reverse-engineering and black-box API tests to identify vulnerabilities in rentable IoT devices. As shown in Figure 4, the steps of our workflow are as follow:

S1: Capture Network Traces and Extract Firmware. Many rentable devices (e.g., EV chargers) are unsafe for hardware tampering because they run on high-voltage household (e.g., 220V) or industrial alternating current (AC) power, not batteries. We disassemble these devices and use bench power supplies to partially boot [58] and run the core chips, such as cellular chips and controller chips (e.g., STM32F4, see Table 2). After booting up the system, we can perform further analysis (e.g., dump logs, or debug) by connecting jumper wires to these chips' pin-outs.

We capture the network traces of these devices to reverse engineer [17, 21, 66] their communication protocols.Due to the high cost of hijacking cellular network communications through fake 4G base stations, we chose to capture the internal network traces of cellular chips, which are not encrypted by the SIM card's cryptographic keys. We find that all cellular chips allow developers to enable the debug log via AT commands [3], which enables us to extract the cellular chips' internal payloads from the logs. By exploiting this vulnerability (**D1** in § 3.3), we can capture the network traces of all target devices (see Table 2) from the pinouts of the cellular chips (see Figure 8). However, *Meituan* bicycle is quickly disconnected by their server and we fail to gain enough logs.

For several devices (e.g., Teld, Starcharge), simply capturing their network traces is not enough because the traces cover only a few commands or the messages are encrypted. We need to debug further and analyze the firmware to extract extra commands and retrieve the hard-coded encryption keys.

Table 2: The physical devices studied in this work and their hardware details. (∅: no such module/feature, -: unknown)

Device Type	Device	#Device Deployed	Main MCU	4G MCU	OS	#Func	Encryption Algorithm	Protocol	#CMD/ Traces	Can Debug	Extract Firmware	Phantom Client
EV Charger	Teld	100k+	STM32F207	EC200N	Baremetal	1884	AES	Binary	68/50	✓	✓	✓
	Starcharge	100k+	LPC1778FBD144	ME3630	FreeRTOS	1030	3DES	Binary	66/63	✓	✓	✓
	Potevio	30k+	GD32F407	N58	μC/OS-II	1325	∅	Binary	56/73	✓	✓	✓
Charger	Xlvren (Charger)	100k+	HC32F460	A7670	ZephyrOS	1718	∅	MQTT	34/79	✓	✓	✓
	Lvcc (Charger)	100k+	FM15F366	CUIoT	Baremetal	711	∅	Binary	42/88	✓	✓	✓
	Xlvren (Cabinet)	100k+	STM32F407	SIM-7600CE	eCos	1899	∅	Binary	46/68	✓	✓	✓
	Lvcc (Cabinet)	100k+	FM15F366	EC200N	Baremetal	726	∅	Binary	36/44	✓	✓	✓
	Lvcc (Socket)	100k+	∅	Air720UH	FreeRTOS	-	∅	Binary	18/60	✗	✗	✓
	JieDian	1,000k+	S34ML02G2	N58	Linux	-	TLS	HTTPS	7/288	✗	✗	✗
Mobility	QiXin	50k+	∅	EC200U	-	1645	∅	Binary	24/40	✓	✓	✓
	MeiTuan	10M+	STM32F401	EC200N	FreeRTOS	-	AES	Binary	2/6	✗	✗	✗
Entertainment	YFLe	10k+	∅	EC100N-CN	-	-	∅	Binary	7/25	✗	✗	✓
	BDT	50k+	∅	SIM-A7670C	-	-	∅	JSON	10/32	✗	✗	✓
	WeiPeng	100k+	∅	Air724UG	FreeRTOS	-	∅	Binary	15/48	✓	✗	✓
Tools	AnSheng	10k+	∅	Air780E	FreeRTOS	-	∅	Binary	8/22	✓	✗	✓
	AnKong	10k+	∅	ML307A	-	-	∅	MQTT	8/28	✗	✗	✓
	WZ-Cloud	5k+	∅	Air724UG	FreeRTOS	-	∅	Binary	6/25	✓	✗	✓

As shown in Table 2, 8 devices usually have no firmware protection (**D2** in § 3.3) and we can easily extract or debug their firmware via the JTAG port. Note that many firmware appear to be the debug versions and the plain text log prints are not removed (**D3** in § 3.3). By examining these log strings, we can easily understand the message dispatching approaches in the binary code and get more commands (CMD in Table 2). Only 4 devices encrypt their messages, and we successfully crack the encryption of 3 devices. The fail case is *JieDian*, as we are unable to analyze their firmware. We successfully debug 11 devices to facilitate protocol reverse engineering.

S2: Re-implement Protocols and Perform API Tests. In real-world scenarios, impersonating a server and sending spoofed 4G messages to devices over the air is exceedingly difficult, as it necessitates exploiting zero-day vulnerabilities in 4G networks [15, 33, 47]. Consequently, our focus shifts to exploring how adversaries can impersonate valid clients and send malicious messages to servers via alternative network connections.

We develop phantom clients [69] that replicate original devices by using the same protocols and reusing their credentials to reconnect with servers. To achieve this goal, we analyze network traces and firmware (detailed in Appendix B) to dissect protocol message sequences, structures, and encryption, and then reimplement these protocols in Python. To ensure the correctness of our phantom clients, we compare their messages with those of real clients and verify the server responses at each protocol step, confirming their functional equivalence.

We successfully develop phantom clients for 15 devices, while failing for *Meituan* and *JieDian* due to insufficient network traces. We then use black-box API testing to determine whether attackers can manipulate device states through spoofed requests. Similar to AuthScope [72], we begin by identifying and altering critical message fields, such as IDs and commands, to probe server vulnerabilities through unauthorized requests. We then examine server responses and check the status of devices and apps to assess whether these crafted requests can successfully manipulate the states of other devices, which indicates potential remote exploits.

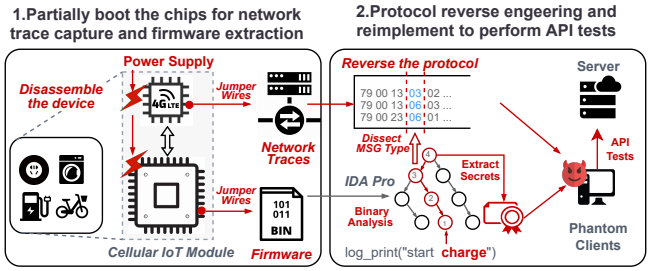


Figure 4: The workflow for studying the rentable IoT device.

Table 3: Device impersonation attacks on local devices.

Device Type	State Spoofing (14)	DoS (5)	Free Use (4)
EV Charger	Teld, Starcharge Potevio	Teld, Starcharge Potevio	Teld
Charger	Lvcc (Socket) Lvcc (Charger), Xlvren (Charger) Lvcc (Cabinet), Xlvren (Cabinet)	Xlvren (Cabinet)	
Mobility	QiXin	QiXin	QiXin, MeiTuan
Entertainment	BDT, WeiPeng		
Tools	AnKong, AnSheng WZ-Cloud		AnKong

4.2 Security Implications in Devices

By testing 15 devices with phantom clients, we identify that 14 devices are vulnerable to device impersonation attacks.

Hazards of Device Impersonation Attacks. The phantom clients can reconnect to the server using the devices' credentials, which can lead to two outcomes: sending spoofed messages to the server to change the devices' states, or replacing the original devices' server connections and preventing them from receiving server messages. With these tactics, we find that 4 devices can be exploited for free use.

(1) Device State Spoofing. Rentable devices typically upload their health states, including cellular signal strength and operational condition (good or faulty), to the server. However, attackers can easily manipulate these states using phantom clients. For example, they can make offline devices appear online, misleading users into paying for unavailable services. Conversely, they can present functional devices as faulty, prompting unnecessary maintenance checks and deterring rentals. We identified 14 devices vulnerable to such state spoofing attacks (shown in Table 3). While some devices attempt to restore their correct states through periodic updates, attackers can override these data more frequently, making the devices continuously display false states. The only exception is *YFLe*, which adds nonce and signature to their messages, and we fail to craft valid messages to change their states.

(2) DoS by Remotely Forcing Devices Offline. For several devices, when we reconnect to the server using the victim device's identifier and credentials, the original devices are forced offline. The server releases the existing connection upon accepting a new connection from the same device. This happens on 5 out of the 15 devices, while the remaining 10 devices maintain both the original connections and the new connections established by our phantom clients. Although the 5 offline devices continue attempting to reconnect to the server, conversely causing the phantom client to disconnect, attackers can persistently attempt to knock the real devices offline, leading to device DoS attacks.

(3) Free Use of Device. Rentable devices are deactivated by server commands; however, attackers can intercept these commands, allowing devices to continue operating beyond the rental period. We identify 3 devices that are vulnerable to this attack. For the *Teld* charger and *QiXin* bicycle, attackers can finalize the rental in the app and then use phantom clients to force the real devices offline. This redirects the device stop commands to the phantom clients instead of the actual devices, enabling free use of the devices. For the *Meituan* bicycle, we can make a local bicycle offline by removing its cellular antenna, which causes the server to process the order as completed normally, but leaves the bicycle unlocked, allowing continued unauthorized use.

For the *AnKong* washing machine, we discovered that clients can publish MQTT messages to each other without restrictions due to the lack of a server MQTT ACL policy [60]. This vulnerability allows attackers to send fake device activation commands to other devices' MQTT topics based on their device IDs. These spoofed messages are then transmitted to the actual devices by the MQTT server, leading to their unauthorized activation. By acquiring device IDs from the devices' QR codes, attackers can freely utilize any device through this attack vector.

Case Study for Attacking Various Devices. All rentable devices share a common architecture managed by their IoT controllers and communicate remotely with the server via a cellular network. However, the analysis and attack details vary across different devices.

(1) Mobility Devices and Stationary Devices. Based on the trust model, existing rentable devices fall into two categories: mobility devices such as e-scooters and e-bikes, which users can take with them, and stationary devices like EV chargers, which are immovable and can only be used at specific locations. Both types of devices offer basic controls for activation and deactivation, and they provide updates on device states, including 4G signal strength (RSSI) and health status. Mobility devices additionally need periodic GPS updates. When devices move outside of approved areas, the server shuts down power, preventing operation beyond designated zones. Attackers may exploit this feature to halt devices using GPS spoofing attacks. Stationary devices are immune to such attacks as they are not restricted by GPS location.

(2) Reverse-Engineering Different Protocols. Except for the 3 devices using MQTT or HTTPS protocols, all others utilize custom binary protocols. Among these devices, EV chargers exhibit a diverse array of commands, including querying the grid status, individual charging channel information (voltage, current, charging status, and energy consumption), controlling the start and stop of charging for specific channels, and exchanging keep-alive messages for both devices and active charging sessions. Each command has a corresponding request and response packet, resulting in a substantial number of distinct command pairs that should be sent in the correct sequence. This extensive command set significantly increases the reverse-engineering effort required for EV charger protocols, making them more difficult to analyze than other devices with simpler protocols, such as e-bicycles and rentable tools, which are typically controlled by simple device start or stop commands. Due to the large number of variable fields in EV charger protocols, extracting and analyzing their firmware from the STM32 MCU is essential to fully understand their protocols.

Note that all of the vulnerabilities discussed in this section can be exploited remotely without touching the physical devices if knowing the victim device's login credentials. Given that many devices share the same hard-coded authentication keys or have no authentication, we further explore the possibility of exploiting more devices on a large scale via ID enumeration attacks in § 6.3.

5 Analyzing Vulnerabilities of Apps

In this section, we perform a semi-automated API black-box test to detect vulnerable app APIs that can exploit other devices or users.

App Collection. We identify IoT companion apps for rentable devices by searching for product names as keywords across various App markets. In China, most devices prefer WeChat mini-programs over native Android or iOS apps, as the convenience of accessing these mini-programs directly within the WeChat platform, which eliminates the need for users to

Table 4: Vulnerabilities in IoT companion apps.

Device Type	Attacking Users		Abusing Devices	
	Privacy Leakage	Account Hijack	Payment Bypass	Manipulate Device
EV Charger	Sunmue, Potevio	Teld		
Charger	LuLuChong, NBLinks ZhouDian			LuLuChong, NBLinks
Mobility	YueHuoCX, YueShiJi GXRongYi	YueHuoCX, YueShiJi GXRongYi,DFPV	QiXin, GO-ON	YueHuoCX, YueShiJi GXRongYi, Hozonauto DFPV, Lime, Helbiz
Tools		AnSheng	HQJL	
Entertainment	BDT		QSMX, Dadaball	

install separate mobile apps. Note that not all of these apps (or mini-programs) can be selected for research since we cannot find valid QR codes for them, which are necessary to rent the devices. As a result, we only find Android apps for 6 Chinese devices. For the devices without mobile apps, we focus on their WeChat mini-programs. We also identify 21 apps that service in countries outside of China, such as Lime e-scooters in the US and Europe. However, we only successfully register for 11 of them. As listed in Appendix A, finally, we get apps for a total of 81 Chinese rentable products (including 75 WeChat mini-programs and 6 Android apps), and 11 iOS apps for Europe/US products.

Analysis Methodology. We focus on identifying payment and API access control vulnerabilities that can result in unauthorized device control or access to other users' resources. We propose a semi-automated approach to test the native Android/iOS apps and WeChat mini-program apps. First, we prepare accounts and login to these apps, which can only be done manually, as rentable IoT apps typically require real-name registration and a deposit before use. While processing the deposit, we manually check if the payment arguments can be manipulated. Then we manually trigger the core logic of the apps by scanning their QR codes and selecting the appropriate UIs to rent and return a physical rentable device. Simultaneously, all requests can be captured by the Burpsuite proxy [2]. After these manual steps, the remaining tasks are black-box API testing, where requests are automatically mutated and replayed in BurpSuite. Specifically, we utilize multiple accounts, with some acting as attackers to gain unauthorized access to the resources of other victim accounts. If the forged requests are accepted by the server (e.g., return HTTP 200), we continue to manually check for access control violations in these APIs. We further validate if these vulnerable APIs can be further exploited by ID enumeration attacks to cause large-scale damage to other devices or users, which is discussed in § 6.4. We discover 34 vulnerabilities in 23 IoT apps, which can be divided into two types based on the victim:

Attacking App Users. As shown in Table 4 (attacking user column), we identify multiple authentications or access control vulnerabilities in mobile apps that allow attackers to steal other users' personal information or hijack others' accounts.

(1) Privacy Leakage. We identified 9 vulnerable IoT apps that leak the users' most sensitive data, such as users' real names, identity card numbers, phone numbers, etc. 8 of the 9 products (excluding *Sunmue*) contain vulnerable APIs that mistakenly return all user fields stored in the same database column, failing to exclude sensitive data from the response, even though not all of these fields are required to be displayed in the UI. Note that *HuoYueCX* E-Bike has added $*$ to mask some sensitive data but still leaks users' home addresses. *Sunmue* provides a super account whose `uid` is 0 and can access all users' orders. By retrieving the order information, we can get other users' phone numbers and vehicle license plate numbers.

(2) Account Hijacking. We identify 6 apps that contain horizontal privilege escalation vulnerabilities, which permit attackers to hijack other user accounts to access devices. For 3 of them, attackers can directly modify the device activation request's *uid* making the charging order to other users' accounts Specifically, *YueHuoCx* possesses a vulnerable API that inadvertently exposes other users' access tokens. For 2 apps, i.e., the *Potevio* EV charger and *DFPV* rentable EV, attackers can steal other users' accounts if they acquire the victim's phone numbers. Similar to previously documented insecurities in face verification system [68], the *Potevio* EV charger app exposes a vulnerable login API that is only locally authenticated with a user's fingerprint or face identification. Once the local biometric authentication is bypassed, attackers can substitute the authenticated phone number with that of the victim, effectively seizing control of the account. The *DFPV* rentable EV app's login API mistakenly returns the SMS one-time password (OTP) [41] to the app and attackers, allowing attackers to gain unauthorized access to any accounts and control them by stealing the SMS verification code. Additionally, *Ankong* app exposes vulnerable device binding APIs, which attackers exploit to elevate their privileges to device administrators using device IDs, thus gaining unfettered access to the devices.

Abusing Devices via Apps. As shown in Table 4 (abusing device column), we identify multiple vulnerabilities that allow attackers to exploit devices via payment bypass or device manipulation APIs.

(1) Payment Bypass for Free Device Usage. We identify 4 apps on WeChat that can manipulate the payment parameters to alter the transaction amounts. For instance, the *QSMX* app, which offers various vehicles, such as the E-Car, E-Boat, and E-Bicycle for sightseeing, enables all orders directly via its WeChat mini-programs. When a user chooses a product, the app communicates the type and associated cost of the product to the server, which then initiates a payment transaction based on the cost data received from the app. By employing Man-in-the-Middle (MitM) attacks, we demonstrate the feasibility of altering payment requests, such as changing ¥10 to ¥0.01, for nearly free renting of these products. This vulnerability also allows attackers to alter the payment of deposits, which can significantly reduce their attack costs. Furthermore, the GO-ON e-scooter iOS app in South America has vulnerabilities in its membership plan API that expose test accounts, allowing free use of any e-scooter.

(2) Manipulating Remote Devices via Apps. We identify 9 apps that can be abused to control remote devices via horizontal privilege escalation. For the 3 e-bikes (includes *HuoYueCX*, *YueShiJi*, and *GXRongYi*) and 2 e-bike chargers (includes *LuLuChong* and *NBLinks*) apps, we can only disable active devices, such as stopping the charging process or immobilizing the bicycles. These apps lack APIs for activating devices, which are instead automatically activated post-payment. Remotely stopping a moving e-bike can cause it to gradually slow down and halt, potentially injuring the user. Note that the remote stop device poses a significantly greater risk to rentable bicycles equipped with locks, as it suddenly locks and immobilizes the bicycles. We can fully control others' devices for the two rentable EV apps, i.e., *Hozonauto* and *DFPV*. Their vehicle control APIs only validate whether the user is valid and whether the vehicle identification number (VIN) is valid, without confirming the user's access to the current vehicle. Attackers can replace the VIN in the requests to activate or deactivate other vehicles illegally.

We identified three vulnerable e-scooter apps operated in Europe/US. The Lime app restricts users to renting or reserving a maximum of five e-scooters via its group ride feature. However, we discovered a vulnerability in the device renting API that allows attackers to bypass this restriction and use more than five scooters. We identified 2 vulnerable trip management APIs in the Helbiz app that enable device manipulation. One allows attackers to retain control over returned devices, and the other permits attackers to alter the state of an unlocked device to locked, while the physical device remains unlocked and usable.

6 Scaling Attack Scope via IDScope

Large numbers of rentable devices in the same type and configuration are deployed and maintained by the same owner, and these devices automatically connect to the vendor's private cloud, which makes them tend to adopt shared keys or weak authentication implementations. As a result, attackers can impersonate these devices and send spoofed requests to the server by manipulating the device ID in vulnerable APIs, leading to large-scale exploitation of users or devices. We propose IDScope to automatically identify and efficiently infer the affected scopes of such large-scale exploitation in the devices and apps of IoT rentable products.

6.1 IDScope Design

Although previous approaches, such as AuthScope [72], can also detect vertical privilege escalation in apps by enumerating resource IDs.Rather than identifying these vulnerabilities, we need to efficiently collect all valid IDs to prove that it is practical for attackers to launch large-scale remote attacks by inferring their device serial number (SN) without getting the

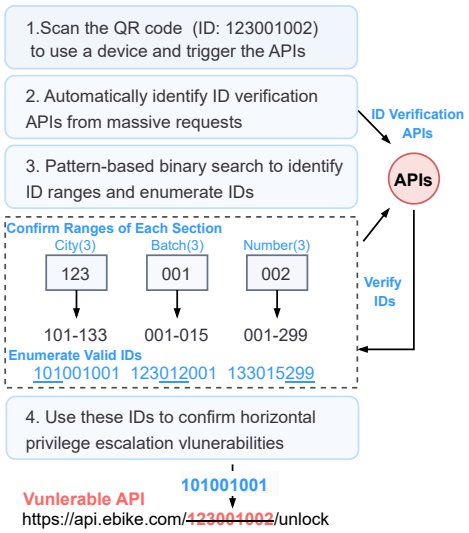


Figure 5: The design of IDScope.

QR code of these devices. However, AuthScope only supports brute-force ID enumeration and is inefficient at inferring IDs.

To fill this gap, we propose an ID enumeration tool called IDScope to quickly infer valid resource IDs (e.g., user ID, device serial number) and assess the affecting scopes of existing authentication, access control, and payment vulnerabilities discussed in § 4.1 and § 5. Figure 5 shows the design of IDScope. The basic idea is to find the ID verification APIs that can validate whether a resource ID is associated with real resources, and validate the guessed ID based on those APIs. It then replaces the IDs in existing requests to visit the vulnerable APIs and check whether the requests with spoofed IDs are accepted by the server. To efficiently guess IDs, IDScope has two key designs:

Automatically Verify Resource IDs. To bootstrap the analysis, we first need to manually scan a QR code to activate and deactivate a device in the rentable IoT apps to trigger device rental requests. By examining these requests, IDScope can automatically identify ID strings and match these strings in various API URLs and parameters. When the ID strings appear in both the payload and response of certain APIs, it means that these APIs may serve as resource verification IDs. We further confirm these APIs by checking keywords (e.g., resource not existing) in their response to exclude the false positive cases. After finding out the ID verification APIs, IDScope can automatically guess new IDs by mutating IDs based on the seed ID (in the QR code) and verifying these IDs using the verification APIs.

Efficiently Enumerating IDs. Since rentable IoT products typically have integer-based resource IDs, employing binary search to guess their ranges is much more efficient than brute-force number enumeration. We observe that device serial numbers often exhibit identifiable patterns, with specific sections representing distinct information. For instance, certain initial digits indicate the device model, while the concluding

Table 5: IDScope's success rate in enumerating different IDs.

#App	Service Area	Device Serial Number		User ID	
		#Enumerable	#Pattern	#Enumerable	#Pattern
81	China	48 (59.2%)	32 (39.5%)	14 (17.3%)	7 (8.6%)
11	Europe/US	8 (72.7%)	6 (54.5%)	- *	1 (9%)

* Fail to enumerate IDs due to not finding ID verification APIs.

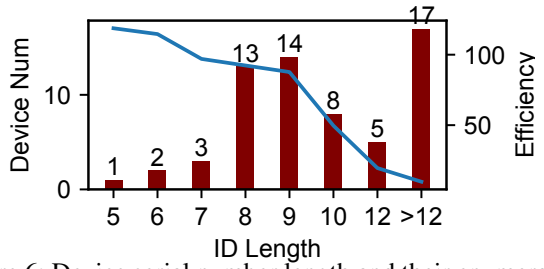


Figure 6: Device serial number length and their enumeration efficiency (how many IDs can be guessed by each search).

digits denote the serial number. IDScope first tries to identify such patterns and confirm the ranges of each section. Then, it applies binary search on each section. This approach can significantly reduce the search space on pattern strings.

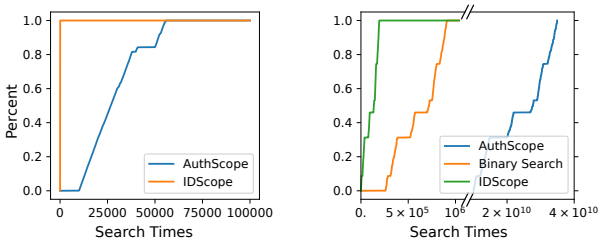
Finally, IDScope verifies whether the existing vulnerable APIs can access the resources with the enumerated IDs. For instance, by replacing the user IDs, IDScope can check if the APIs can access the profiles of other users. APIs with horizontal privilege escalation flaws [45] can potentially be exploited to target all users or devices if attackers can enumerate all IDs. In such cases, rate-limiting measures are ineffective in blocking the attackers, as they can easily switch between different user IDs.

IDScope Implementation. We prototype IDScope as a BurpSuite [2] plugin in about 1200 lines of Python code. Note that IDScope can also work as a standalone Python program to off-path send requests to the servers by reusing the access tokens. IDScope sends multiple requests to quickly infer the valid ranges of ID values. These requests are sent in a single PC at a very slow frequency and do not lead to a DoS attack. We empirically specify several ID patterns and first check whether the ID follows these patterns. For the IDs that do not match any pattern, we cannot use the pattern-based binary search and fall back to the normal binary search.

6.2 The Effectiveness of IDScope

We evaluate the effectiveness and efficiency of IDScope using 75 rentable IoT apps collected in § 5.

Experiment Setup. We run IDScope on real products and ID datasets granted by the vendors. We first check how many rentable IoT apps are successfully processed by IDScope to correctly identify the ID verification APIs and fully enumerate the valid ID ranges of real devices or users. Rather than dump all their device IDs, we get a precise range of these devices as the IDs of these devices are usually continuously incremented.



(a) IDScope vs Brute Force on LuLuChong Dataset (5 digits)

(b) IDScope vs Binary Search on QSMX Dataset (11 digits)

Figure 7: The CDF of enumerated IDs and search times.

IDScope can further precisely verify if each number in this range is a real device. We only do this under the grant of the vendors. Furthermore, we obtain several device ID datasets from the vendors to evaluate the performance of IDScope.

Enumeration Results. As presented in Table 5, IDScope succeeds in enumerating the device serial number (SN) of 56 products, and the rest 36 products' SN are too long and lack discernible patterns, making enumeration impossible. However, some of these apps leak their SN in the map APIs, enabling attackers to easily steal all the device SNs (§ 6.3). Specifically, IDScope successfully identifies the ID patterns for the device serial numbers of 28 products. IDScope successfully enumerates user IDs for 14 products only, as most lack accessible user ID verification APIs, requiring both an access token and the user ID to construct valid requests.

Enumeration Performance. Figure 6 shows the enumeration efficiency of IDScope on products with different ID lengths. Figure 7 shows a comparison of different enumeration approaches. IDScope's pattern-based binary search approach is 4 magnitudes faster than AuthScope's brute-force approach for products with sequential IDs. For IDs longer than 6 (e.g., *QSMX* adopts 11 digits with patterns), AuthScope cannot infer their ranges within a reasonable timeframe. While naive binary search could be employed, it would require sending tens of thousands of requests, taking several hours to complete. IDScope can achieve a performance improvement of 3 to 4 orders of magnitude over naive binary search and is 10 orders of magnitude faster than AuthScope. Consequently, IDScope can identify all devices of a vendor with much fewer requests.

6.3 Scale Device Side Exploits via IDScope

In § 4.2, we identify 14 devices that are vulnerable to device impersonation attacks. Since some of them lack secure authentication and attackers can impersonate them using their device IDs, we further leverage IDScope to validate whether these vulnerabilities can be exploited at scale to attack all devices via ID enumeration attacks.

Large-Scale Obtaining Device ID. We identify two attacks that can lead to large-scale device ID leakage. As shown in

Table 6: Device vulnerabilities that can be exploited at scale.

Device Type	Device	Obtain ID	State Spoofing	DoS	Free Use
EV Charger	Teld	A1 A2	✓	✓	✓
	Starcharge	A1 A2	✓	✓	
	Potevio	A2	✓	✓	
Charger	Xlvren(Charger)	A1 A2	✓		
	Lvcc(Charger)	A1 A2	✓		
	Xlvren(Cabinet)	A1 A2	✓	✓	
	Lvcc(Cabinet)	A1 A2	✓		
	Lvcc(Socket)	A1 A2	✓		
Mobility	QiXin	A2	✓	✓	✓
Entertainment	BDT	A1	✓		
Tools	AnKong	A1	✓		✓
	AnSheng	A1	✓		
	WZ-Cloud	A1	✓		

Table 7: App vulnerabilities that can be exploited at scale

Device Type	Vendor	ID Emueration		Large-Scale Privacy Leakage	Large-Scale Device Free Usage		
		Uid (Len)	SN (Len)		Account Hijack	Payment Bypass	Manipulate Devices
EV Charger	Teld	✗(14)	✗(13)		✓ [1]		
	Sunmue	✗(15)	✓(10)	✓ [1]			
Charger	LuLuChong	✓(11)	✓(5)	✓			✓
	NBLinks	✓(9)	✓(9)	✓			✓
	ZhouDian	✓(7)	✓(13)	✓			
Mobility	YueHuoCX	✓(7)	✓(10)	✓	✓		✓
	GXRongYi	✗(24)	✓(9)				✓
	YueShiJi	✗(24)	✓(9)				✓
	Hozonauto	✗(11)	✓(17)				✓
	DFPV	✗(11)	✓(17)				✓
	QiXin	✓(7)	✗(12)			✓ [2]	
	GO-ON	✗(16)	✓(6)			✓	
Tools	HQJL	✓(7)	✓(8)			✓	
	Ansheng	✓(9)	✓(9)		✓		
	Dadaball	✓(5)	✓(8)			✓	
Entertainment	QSMX	✓(32)	✓(11)			✓	
	BDT	✓(6)	✗(16)	✓			

[1] Besides enumerate IDs, attackers can exploit the super accounts (vertical privilege escalation).
[2] *QiXin*'s device SN cannot be enumerated and is leaked by map API.

Table 2, we can get the ID of every device for 13 products.

(1) Enumerating Weak ID with IDScope (A1). Some devices reuse the device serial numbers (in QR codes) as device IDs in the firmware. We obtain these devices' serial numbers by applying IDScope on their companion apps. These enumerated IDs can be abused to exploit vulnerable APIs.

(2) Exploiting the App API to Leak Device ID (A2). Rentable IoT products need to display the user's nearby devices in their apps. To implement this feature, the server provides device profile retrieval APIs to the app, which fetches nearby devices and displays them on the map using these APIs. We identify multiple vulnerable app APIs that mistakenly return all of a device's fields reading from the databases, including sensitive data like the device ID, while only a few of these fields are actually displayed in the app. By modifying the GPS location, we can abuse these APIs to leak the IDs of all devices.

Remotely Exploiting Devices at Scale. As shown in Table 6, for 13 of the 14 vulnerable devices, attackers can replace the device ID to connect to their server and perform various attacks. These devices use weak authentication, and attackers can easily bypass authentication and authorization and send spoofed requests to affect other devices. Specifically, 3 of them hard-coded common keys in all their devices, and 10 devices do not authenticate the devices. Theoretically, attackers can get all device IDs and remotely exploit any device. For *MeiTuan* bicycle, we fail to reverse their protocol and do not identify any APIs with access control vulnerabilities. *WeiPeng* uses unique keys for each device. As a result, we can only exploit their vulnerabilities on local devices.

6.4 Scale App Side Exploits via IDScope

We adopt IDScope to confirm if the app vulnerabilities discussed in § 5 can lead to the large-scale exploitation of users or devices. The results are shown in Table 7.

User ID enumeration. IDScope succeeds in enumerating all the user IDs of 9 products, which with ID length of 5 to 11. In particular, IDScope can identify the ID pattern of *LuLuChong* and *NBLinks*, which contain common alignment prefixes making it easy to find out all valid user IDs. Using only 7 digits as user IDs seems inadequate for accommodating the increasing number of users in two products. For example, *YueHuoCX* [11] has millions of users but relies on just 7 integers for user identification.

The failed cases include two devices that use the phone number (11-digit) as user IDs, which are inefficient to be enumerated by IDScope. However, for popular products, the phone numbers may still be enumerated, as attackers can get hits by matching the rentable IoT deployment cities with the guessed phone number's registration cities. The other 4 products utilize IDs consisting of 14 to 24 irregular numbers or strings, which can resist the enumeration of IDScope.

Device ID enumeration. IDScope can efficiently guess 15 vulnerable devices' serial numbers and identify their ID patterns. For example, the *QSMX* rentable e-bike/boat uses 9-digit IDs, but they have a pattern of 3-4, where the first 3 digits refer to the city code and the last 4 digits are the device numbers. Attackers can easily infer the valid ranges of these 3-digit sections. The VIN number also has a similar structure [9] that cars of the same vendors only vary in some sections of the whole string. The failed cases (e.g., *QiXin*) are devices that use a long random string containing both numbers and characters.

Large-Scale Exploitation on Users and Devices. For 9 products susceptible to user ID enumeration, attackers can exploit vulnerable APIs using the victim users' IDs, leading to large-scale privacy leakage and unauthorized access to private user data. In addition, five products exhibit account hijacking vulnerabilities, enabling attackers to seize control of user accounts on a large scale. Once these accounts are compromised, attackers are positioned to manipulate associated devices via their serial numbers. For 5 products with payment bypass vulnerabilities and 7 products with device manipulation vulnerabilities, attackers leverage enumerated device SNs to access or rent devices without cost, leading to extensive unauthorized usage. Notably, the `Teld` EV-charger

has another vulnerable API allowing users to be added to a free-charge whitelist, which can be exploited to enable free charging on any device after gaining device SNs from the app's map APIs.

Although the 3 vulnerable e-scooters (discussed in § 5) apps' device SN can all be enumerated, Lime and Helbiz still require attackers to pay for their orders, while GO-ON allows attackers to large-scale unlock all e-scooters for free. Combined with credit card fraud, attackers can unlock unlimited Lime e-scooters and manipulate all devices of Helbiz.

7 Responsible Disclosure and Mitigation

7.1 Responsible Disclosure

From 2022 to 2023, we identify 28 vulnerabilities in 19 apps and 18 vulnerabilities in 11 devices of rentable products. We inform vendors of our findings via email, phone, or WeChat, with communications typically lasting one to two months. All vulnerabilities are confirmed by the vendors, resulting in 15 CVE/CNNVD entries and 3 NVDB entries. The vendors promptly resolve the 28 vulnerabilities in apps. The 18 vulnerabilities on devices are addressed in their new devices but some (e.g., hardcoded keys) remain in their legacy devices.

In 2024, we investigate 6 new Chinese rentable devices (including *AnSheng*, *AnKong*, *WeiPeng*, *YFLe*, and *BDT*) and find 5 vulnerabilities across 3 physical devices and 2 vulnerabilities within two apps from 3 different vendors. To date, only *Ankong* has confirmed its 3 vulnerabilities and we are continuing discussions with the remaining 2 vendors. In addition, for the 11 e-scooter/e-bike operating in other countries, we discovered 4 vulnerabilities in the iOS apps of Lime, GO-ON and Helbiz. We inform the affected vendors of the vulnerabilities via email. They have acknowledged forwarding the information to their technology teams for further confirmation. We track vendor feedback online at https://vehicle-security.github.io.

7.2 Mitigating ID Enumeration Attack

The root cause of ID enumeration attacks is that some APIs lack proper access control, and attackers can illegally access unauthorized resources via the resource ID. Before being detected by the server, they can attack as many devices as possible by large-scale enumerating the resource IDs. It is impossible to eliminate all these vulnerabilities to make ID enumeration harmless. A straightforward way to mitigate this attack is to append random characters to the ID string and increase its length to make it much harder to enumerate. However, some products need to retain a simple ID to make it easier for users to manually type in the ID if the QR code is broken. Also, some resources may use the auto-incrementing fields of the database (e.g. primary key) as IDs, which are expensive to change due to the complexity of the backend. Simply adopting a rate limit is also inadequate as the apps may also frequently invoke these APIs to query devices via their IDs.

To enhance the security of using short digits as IDs, we propose a decoy ID-based solution [37] to quickly detect the ID enumeration attack. The basic idea is to mix the real IDs with fake IDs, and the fake IDs are not actually used by the devices, but can only be accessed by attackers who try to craft requests to enumerate IDs. Apps typically do not access these fake IDs unless users mistakenly input incorrect IDs, which occurs at a very low rate. If a client consistently triggers fake IDs at a high frequency, it strongly indicates the adversaries are attempting to enumerate IDs. The server can then block these clients.

Security Analysis. We assume that attackers can change their IPs and user IDs to bypass the rate limit. The defense goal is to prevent them from exploiting the devices with continuous serial numbers. We choose to shuffle the real IDs and mix them uniformly with the decoy ID. By studying the ID distribution in real products, we notice that the real device numbers are about 5% to 45% of all the ID space. That is, for a 5-digit ID, there can be 5000 (5%) to 45K (45%) consecutive device numbers ranging from 0 to 5000/45000. This allows us to insert 3 to 8 fake IDs in every 10 IDs, which is enough to prevent enumeration in any range of device numbers. By setting a threshold of three fake IDs that a malicious user can trigger, attackers can be detected within 10 enumeration requests.

7.3 Mitigating Other Vulnerabilities

We correspond with multiple vendors via email to facilitate the mitigation of vulnerabilities.

(1) Mitigate Physical Devices' Threats. Local exploits, although limited in scope, typically begin with attackers analyzing the hardware of devices by extracting firmware [24], injecting intra-board messages [3], and creating phantom clients [69]. Such methods require disassembling the devices and exploiting weaknesses in the MCUs [58]. To counter these tactics, vendors are advised to redesign the physical structures of their products to hinder disassembly and reassembly efforts, enhance the security of MCU firmware [5], and integrate anti-tamper mechanisms that can detect any disassembly attempts [57]. One plausible measure to protect MCU firmware involves implementing one-time programmable eFuses [6], which can permanently disable JTAG and SWD interfaces, thus securing the hardware from unauthorized modifications. Additionally, installing hardware that monitors and records the device's internal status [57] can help servers identify hardware tampering or falsified device states. However, these hardware updates can be extremely time-consuming due to the vast number of devices involved, presenting a significant challenge in large-scale implementations. Therefore, we suggest vendors deploy server abnormal detection [43] for frequent invalid requests to identify and ban compromised devices.

Currently, only the *Meituan* shared bicycle uses special physical structures to automatically disconnect devices when they are disassembled.

*(2) Mitigate the Device-Side Remote Threats.*Among 15 remotely exploitable products analyzed, only 2 employed message encryption, sharing keys across devices, while the others lacked encryption or robust authentication methods. Device IDs, often simplistic and predictable, were consistently exposed via app APIs, facilitating potential large-scale attacks [24]. Notably, in our work five of these devices employed easily guessable, sequential IDs, increasing their vulnerability to enumeration attacks.

While some vendors have begun issuing firmware updates since October 2022 to implement unique device passwords, significant challenges persist, especially for products like *LVCC* that lack support for OTA updates. Recommended mitigation strategies include the implementation of TLS for secure communication, the use of unique device passwords, the deployment of session-specific tokens, and the strengthening of ID security measures to enhance overall device-side protection. Although plaintext passwords can still be extracted from firmware, large-scale remote exploitation is impractical as attackers can only access limited local devices.

*(3) Mitigate the App-Side Remote Threats.*Companion apps often handle payment or device control requests, which can be compromised by attackers due to insecure implementations. This includes vulnerable server APIs that fail to properly manage authorization or inadvertently expose sensitive data [72]. Such vulnerabilities pose a risk to a vast number of devices. Vendors have responded swiftly by rectifying these flaws. Furthermore, they are enhancing the security of device IDs by lengthening them, incorporating decoy IDs, and implementing abnormal activity detection systems.

8 Related Work

EV Charger and E-Scooter Security. Our study includes nearly 40% of the devices that are rentable chargers for e-bikes or EVs. We specifically investigate the security vulnerabilities in the firmware and apps of these rentable devices. While previous research has addressed physical layer security [14] and communication security within charging standard protocols [23], these areas are orthogonal to our focus, which centers on issues arising from the shared nature of these devices. Previous work [59] focuses on the privacy issues in e-scooter rental apps, we systematically study the exploitation of their apps and devices.

Enumeration Attacks. Existing works focus on detecting enumeration attacks on cloud environments, usernames, etc. [12,28] Enumeration attacks specifically targeting IoT devices remain unexplored. In our work, we uncover and demonstrate novel types of enumeration attacks that exploit various resource IDs of rentable IoT devices, presenting unique challenges and requiring distinct enumeration strategies tailored to the rentable device context.

IoT Access Control. Multiple users and devices [55,56] can lead to significant authorization and usage coordination challenges in a smart home ecosystem. Kratos+ [54,55] addresses these challenges by implementing a system of partial authorization, which ensures normal operation while mitigating the risks associated with excessive permissions, which not only maintains normal operations but also prevents the risks of excessive permissions that could lead to device tampering. To further enhance the security of the Smart Home System (SHS), Aegis framework [13, 52, 53] supports various network communication protocols and enhances threat detection capabilities. It effectively monitors both encrypted and unencrypted traffic, identifies potential threats from integrated sensors in-home devices, and deploys defensive strategies to mitigate these vulnerabilities. However, it still needs to explore how to adapt these solutions to rentable IoT devices as they have a different backend architecture.

IoT Authentication Security. Existing works [18,34,35,51, 69] mainly focus on the interactions between smart home IoT devices and clouds. rentable IoT devices represent new research problems as they communicate directly with the cloud, while smart home devices may use the companion app [46,70] as gateways or IoT hubs [69] to connect to the IoT cloud. The MITM attacks (e.g., OTA hijack) on smart homes devices [18, 70] cannot be applied to cellular IoT devices as their communication is over the LTE cellular network. Similarly, the large-scale weak authentication found in cellular IoT devices by our study is unlikely to happen in smart homes, which are usually managed separately by users [34,35]. Note that there are also some approaches [26, 64] that propose new mechanisms for securing IoT authentication, which can help to mitigate the weak authentication problem and device impersonation attacks discussed in our work.

IoT Reverse Engineering. Reverse engineering is widely used [18, 62, 69] for researching the risks in closed-source IoT products. Our work focuses on reversing the customized binary protocols of cellular IoT devices. The existing protocol reverse engineering efforts can be divided into two categories: binary analysis based approaches [22,25,42] which dynamically or statically analyze the binary code and memory data of protocol processing, and network message traces based approaches [38,39,66] which distinguish the message types and identify their formats by message cluster and alignment. Only adopting firmware analysis [22] is inadequate for retrieving message formats due to their complex, variable-length parameters. The challenge in reversing cellular IoT device protocols lies in understanding the semantic of all commands, not just clustering messages [66] to identify format, as network traces can only cover a few commands of the devices. We address this problem by analyzing the firmware to extract new commands from the strings of the log print code.

IoT Firmware Security Extensive research efforts are pro-

posed to identify the vulnerabilities [27, 29, 44, 63] or enhance the security in IoT devices [31, 49]. HEAPSTER [29] discovers the vulnerable heap allocators in IoT firmware. DICE [44], P2IM [27], and Jetset [36] improve the firmware rehosting [30] to facilitate the firmware fuzzing test. Our work do not exploit the vulnerabilities in firmware but extract secrets from them. HERA [49] and RapidPatch [31] address the challenge of hotpatching real-time IoT devices for timely bug fixes. These works are orthogonal to our study.

IoT Cloud and Companion App Security. Existing works focus on smart homes' cloud access control security [35, 69] and protocol security [34, 60]. Jia et al. [34] and MPInspector [60] focus on MQTT based IoT devices, while most cellular IoT devices use customized binary protocols. Multiple works [46, 50, 61, 70] choose to study IoT security from the perspective of the companion apps. MaaG [70] identify the vulnerable access controls between IoT devices and their companion apps. BLESCOPE [71] discovers attackers can fingerprint the BLE based IoT devices by obtaining their static UUIDs from the companion apps. DIANE [50] analyzes the companion apps to efficiently generate valid inputs to fuzz IoT firmware. Our study reveals new consequences of large-scale exploiting the users and devices by abusing app APIs.

9 Discussion

We talk to vendors to identify the root causes of their vulnerabilities and learn from their mistakes to establish best practices for securing rentable IoT devices.

The Root Causes of Using Weak IDs and Common Keys. As discussed in § 2, rentable IoT products commonly utilize three resource IDs: the user ID, the device serial number, and the device ID. Vendors choose simple IDs like auto-incrementing user IDs for their easy implementation and human readability, ignoring the security risks, which have been underreported until now. Using common keys is also a trade-off in device manufacturing. The device serial number is often printed on the physical device to allow manual entry, favoring shorter IDs for ease of use. Additionally, because of the large number of devices, to streamline device provisioning and communication processes, vendors often employ common keys across multiple devices and use continuous integers as serial numbers.

Limitation and Future Works. Our work has two primary limitations, and we leave them for future work.

(1) Investigating More IoT Products. Two key factors restrict the number of products studied. First, it is extremely expensive for us to get more devices or apps for research. We can only buy very few devices as a limited number of vendors sell shared IoT devices to individual operators. It is also time-consuming to obtain devices from the vendors by cooperating with them. However, attackers face little hindrance in acquiring these devices, as they can illegally obtain (e.g., steal or rob) a device, given the huge number of devices operating unmanned. Furthermore, accessing the QR codes of rentable IoT apps presents another challenge, as they are typically only available on physical devices deployed across various cities. It is interesting to extensively study all the components of rentable IoT devices, such as EV charger's management website [48] and charging standard [40].

Second, testing rentable IoT products is labor intensive. Many rentable IoT apps adopt real-name registration and need to pay a deposit before use. These processes cannot be automated now. Reverse engineering IoT firmware and Android apps also require sophisticated hacking skills. In the future, we plan to study a wider range of rentable IoT products in a more efficient way.

(2) In-depth Study of the Cellular Network and Chips. We believe that the vulnerabilities outlined in this paper may also be present in other cellular IoT devices. Unfortunately, our attempts to crack the secure boot of cellular MCUs have been unsuccessful, hindering further exploration of their firmware. Additionally, the absence of professional cellular network security analysis tools and environments, which are often costly and must be operated within strictly controlled conditions as mandated by laws in many countries, including China, has limited our ability to delve deeper into cellular network security. This restricts our capability to experiment with and analyze cellular networks safely and legally. If attackers can intercept the 4G/5G cellular network, they can use our vulnerabilities to manipulate multiple physical devices by forging server commands.

10 Conclusion

In this paper, we conduct the first study to introduce the architecture and uncover the security implications of rentable IoT products. By investigating 17 physical devices and 92 IoT apps, we find that rentable IoT devices exhibit multiple common design and implementation vulnerabilities in both their local devices and companion apps. To validate these findings in real products, we first propose a semi-automated approach to test the APIs of rentable IoT devices and apps, successfully identifying 23 vulnerabilities across 14 physical devices and 34 vulnerabilities within 23 apps, which can affect 28 products. Moreover, we also find these devices utilize weak IDs across multiple resources, posing a risk of large-scale exploitation to users or devices. To assess the scope of affected devices, we then propose an automated ID enumeration detection tool, called IDScope, which confirms that 13 devices and 17 apps are vulnerable to large-scale attacks that can affect all users or devices from the same vendor for 24 products. Our work elucidates the root causes of these vulnerabilities, enabling effective vendor mitigation and resulting in 18 CVE/CNNVD/NVDB numbers.

Acknowledgment

We would like to thank our shepherd and the anonymous reviewers for their valuable comments. This work is in part supported by the National Natural Science Foundation of China (62272265, 62132011, and U1936121). Qi Li and Jianwei Zhuge are the corresponding authors of the paper.

References

[1] Bug Bounty Plan for Lime E-Scooter. https://bugcrowd.com/lime.

[2] BurpSuite. https://portswigger.net/burp.

[3] Enable cellular chip's logs. https://doc.openluat.com/wiki/29?wiki_page_id=3371.

[4] Potevio Charger for vehicle. http://www.evcard.ptne.cn.

[5] Protect MCU firmware. https://hackmag.com/security/protec-stm32/.

[6] Secure JTAG. https://www.digi.com/resources/documentation/digidocs/90001546/concept/trustfence/c_secure_jtag.htm.

[7] Starcharge Charger for vehicle. https://www.starcharge.com/en.

[8] Teld Charger for vehicle. https://www.teld.cn.

[9] The ISO 3779 standard of Vehicle identification number. https://en.wikipedia.org/wiki/Vehicle_identification_number.

[10] The rise and fall of OFO. https://www.scmp.com/tech/start-ups/article/3114932/rise-and-fall-mobike-and-ofo-chinas-bike-sharing-twin-stars.

[11] The website of YueHuoChuXing shared E-Bike. http://yuehuocx.com/.

[12] What are User Enumeration Attacks? https://www.virtuesecurity.com/kb/username-enumeration/.

[13] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. Peek-a-boo: i see your smart home activities, even encrypted! WiSec, 2020.

[14] Richard Baker and Ivan Martinovic. Losing the car keys: Wireless phy-layer insecurity in EV charging. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security*, 2019.

[15] David A. Basin, Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5g authentication. In *CCS*, 2018.

[16] Iulia Bastys, Musard Balliu, and Andrei Sabelfeld. If this then what?: Controlling flows in iot apps. In *CCS*, 2018.

[17] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. M. Munafo. Automatic protocol field inference for deeper protocol understanding. In *IFIP Networking*, 2015.

[18] Marco Casagrande, Eleonora Losiouk, Mauro Conti, Mathias Payer, and Daniele Antonioli. Breakmi: Reversing, exploiting and fixing xiaomi fitness tracking ecosystem. *CHES*, 2022.

[19] Z. Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick D. McDaniel, and A. Selcuk Uluagac. Sensitive information tracking in commodity iot. In *USENIX Security*, 2018.

[20] Z. Berkay Celik, Gang Tan, and Patrick Mcdaniel. Iotguard: Dynamic enforcement of security and safety policy in commodity iot. *NDSS*, 2019.

[21] Jared Chandler, Adam Wick, and Kathleen Fisher. Binaryinferno: A semantic-driven approach to field inference for binary message formats. In *NDSS*, 2023.

[22] Lucian Cojocar, Jonas Zaddach, Roel Verdult, Herbert Bos, Aurélien Francillon, and Davide Balzarotti. Pie: Parser identification in embedded systems. In *ACSAC*, 2015.

[23] Mauro Conti, Denis Donadel, Radha Poovendran, and Federico Turrin. Evexchange: A relay attack on electric vehicle charging system. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS*, 2022.

[24] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A large-scale analysis of the security of embedded firmwares. In *USENIX Security*, 2014.

[25] Weidong Cui, Jayanthkumar Kannan, and Helen J. Wang. Discoverer: Automatic protocol reverse engineering from network traces. In *USENIX, Security*, 2007.

[26] Jingwen Fan, Yi He, Bo Tang, Qi Li, and Ravi Sandhu. Ruledger: Ensuring execution integrity in trigger-action iot platforms. In *INFOCOM*, 2021.

[27] Bo Feng, Alejandro Mera, and Long Lu. P2IM: scalable and hardware-independent firmware testing via automatic peripheral interface modeling. In *USENIX Security*, 2020.

[28] Samira Eisaloo Gharghasheh and Tim Steinbach. Detection of enumeration attacks in cloud environments using infrastructure log data. *Handbook of Big Data Analytics and Forensics*, pages 41–52, 2022.

[29] Fabio Gritti, Fabio Pagani, Ilya Grishchenko, Lukas Dresel, Nilo Redini, Christopher Kruegel, and Giovanni Vigna. HEAPSTER: analyzing the security of dynamic allocators for monolithic firmware images. In *IEEE S&P*, 2022.

[30] Eric Gustafson, Marius Muench, Chad Spensky, Nilo Redini, Aravind Machiry, Yanick Fratantonio, Davide Balzarotti, Aurélien Francillon, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna. Toward the analysis of embedded firmware through automated re-hosting. In *RAID*, 2019.

[31] Yi He, Zhenhua Zou, Kun Sun, Zhuotao Liu, Ke Xu, Qian Wang, Chao Shen, Zhi Wang, and Qi Li. RapidPatch: Firmware hotpatching for Real-Time embedded devices. In *USENIX Security*, 2022.

[32] Syed Rafiul Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. Lteinspector: A systematic approach for adversarial testing of 4g lte. In *NDSS*, 2018.

[33] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 5greasoner: A property-directed security and privacy analysis framework for 5g cellular network protocol. In *CCS*, 2019.

[34] Yan Jia, Luyi Xing, Yuhang Mao, Dongfang Zhao, XiaoFeng Wang, Shangru Zhao, and Yuqing Zhang. Burglars' iot paradise: Understanding and mitigating security risks of general messaging protocols on iot clouds. In *IEEE S&P*, 2020.

[35] Yan Jia, Bin Yuan, Luyi Xing, Dongfang Zhao, Yifan Zhang, XiaoFeng Wang, Yijing Liu, Kaimin Zheng, Peyton Crnjak, Yuqing Zhang, Deqing Zou, and Hai Jin. Who's in control? on security risks of disjointed iot device management channels. CCS '21, 2021.

[36] Evan Johnson, Maxwell Bland, Yifei Zhu, Joshua Mason, Stephen Checkoway, Stefan Savage, and Kirill Levchenko. Jetset: Targeted firmware rehosting for embedded systems. In *USENIX Security*, 2021.

[37] Ari Juels and Ronald L. Rivest. Honeywords: Making password-cracking detectable. In *CCS*, 2013.

[38] Stephan Kleber, Frank Kargl, Milan State, and Matthias Hollick. Network message field type clustering for reverse engineering of unknown binary protocols. In *DSN Workshops (DSN-W)*, 2022.

[39] Stephan Kleber, Henning Kopp, and Frank Kargl. NEMESYS: network message syntax reverse engineering by analysis of the intrinsic structure of individual messages. In *WOOT*, 2018.

[40] Sebastian Köhler, Richard Baker, Martin Strohmeier, and Ivan Martinovici. Brokenwire: Wireless disruption of ccs electric vehicle charging. In *NDSS*, 2023.

[41] Zeyu Lei, Yuhong Nan, Yanick Fratantonio, Antonio Bianchi, and Cisco Talos. On the insecurity of sms one-time password messages against local attackers in modern mobile devices. In *NDSS*, 2021.

[42] Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *NDSS*, 2008.

[43] Yongxin Liu, Jian Wang, Jianqiang Li, Shuteng Niu, and Houbing Song. Machine learning for the detection and identification of internet of things devices: A survey. *IEEE Internet of Things Journal*, 2022.

[44] Alejandro Mera, Bo Feng, Long Lu, and Engin Kirda. Dice: Automatic emulation of dma input channels for dynamic firmware analysis. In *IEEE S&P*, 2021.

[45] Maliheh Monshizadeh, Prasad Naldurg, and V. N. Venkatakrishnan. Mace: Detecting privilege escalation vulnerabilities in web applications. In *CCS*, 2014.

[46] Yuhong Nan, Xueqiang Wang, Luyi Xing, Xiaojing Liao, Ruoyu Wu, Jianliang Wu, Yifan Zhang, and XiaoFeng Wang. Are you spying on me? large-scale analysis on iot data exposure through companion apps. In *USENIX Security*, 2023.

[47] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, Feng Qian, and Zhi-Li Zhang. A variegated look at 5g in the wild: performance, power, and qoe implications. In *SIGCOMM*, 2021.

[48] Tony Nasr, Sadegh Torabi, Elias Bou-Harb, Claude Fachkha, and Chadi M. Assi. Chargeprint: A framework for internet-scale discovery and security analysis of ev charging management systems. In *NDSS*, 2023.

[49] Christian Niesler, Sebastian Surminski, and Lucas Davi. HERA: hotpatching of embedded real-time applications. In *NDSS*, 2021.

[50] Nilo Redini, Andrea Continella, Dipanjan Das, Giulio De Pasquale, Noah Spahn, Aravind Machiry, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. Diane: Identifying fuzzing triggers in apps to generate

under-constrained inputs for iot devices. In *IEEE S&P*, 2021.

[51] Hetian Shi, Yi He, Qing Wang, Jianwei Zhuge, Qi Li, and Xin Liu. Laser-based command injection attacks on voice-controlled microphone arrays. *IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES)*, 2024.

[52] Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac. 6thSense: A context-aware sensor-based attack detector for smart devices. In *USENIX Security*, 2017.

[53] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A. Selcuk Uluagac. Aegis: a context-aware security framework for smart home systems. In *ACSAC*, 2019.

[54] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac. Kratos: Multi-user multi-device-aware access control system for the smart home. In *WiSec*, 2020.

[55] Amit Kumar Sikder, Leonardo Babun, Z. Berkay Celik, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A. Selcuk Uluagac. Who's controlling my device? multi-user multi-device-aware access control system for shared smart home environment. *ACM Trans. Internet Things*, 2022.

[56] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A. Selcuk Uluagac. A survey on sensor-based threats and attacks to smart devices and applications. *IEEE Communications Surveys & Tutorials*, 2021.

[57] Paul Staat, Johannes Tobisch, Christian T. Zenger, and Christof Paar. Anti-tamper radio: System-level tamper detection for computing systems. *IEEE S&P*, 2021.

[58] Sebastian Vasile, David F. Oswald, and Tom Chothia. Breaking all the things - a systematic survey of firmware extraction techniques for iot devices. In *CARDIS*, 2018.

[59] Nisha Vinayaga-Sureshkanth, Raveen Wijewickrama, Anindya Maiti, and Murtuza Jadliwala. An investigative study on the privacy implications of mobile e-scooter rental apps. In *WiSec*, 2022.

[60] Qinying Wang, Shouling Ji, Yuan Tian, Xuhong Zhang, Binbin Zhao, Yuhong Kan, Zhaowei Lin, Changting Lin, Shuiguang Deng, Alex X. Liu, and Raheem Beyah. Mpinspector: A systematic and automatic approach for evaluating the security of iot messaging protocols. In *USENIX Security*, 2021.

[61] Xueqiang Wang, Yuqiong Sun, Susanta Nanda, and XiaoFeng Wang. Looking from the mirror: Evaluating iot device security through mobile companion apps. In *USENIX Security*, 2019.

[62] Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Firmxray: Detecting bluetooth link layer vulnerabilities from bare-metal firmware. In *CCS*, 2020.

[63] Yuhao Wu, Jinwen Wang, Yujie Wang, Shixuan Zhai, Zihan Li, Yi He, Kun Sun, Qi Li, and Ning Zhang. Your firmware has arrived: A study of firmware update vulnerabilities. In *USENIX Security*, 2024.

[64] Yue Xiao, Yi He, Xiaoli Zhang, Qian Wang, Renjie Xie, Kun Sun, Ke Xu, and Qi Li. From hardware fingerprint to access token: Enhancing the authentication on iot devices. In *NDSS*, 2024.

[65] Hojoon Yang, Sangwook Bae, Mincheol Son, Hongil Kim, Song Min Kim, and Yongdae Kim. Hiding in plain signal: Physical signal overshadowing attack on LTE. In *USENIX Security*, 2019.

[66] Yapeng Ye, Zhuo Zhang, Fei Wang, Xiangyu Zhang, and Dongyan Xu. Netplier: Probabilistic network protocol reverse engineering from message traces. In *NDSS*, 2021.

[67] Bin Yuan, Yan Jia, Luyi Xing, Dongfang Zhao, XiaoFeng Wang, Deqing Zou, Hai Jin, and Yuqing Zhang. Shattered chain of trust: Understanding security risks in cross-cloud iot access delegation. In *USENIX Security*, 2020.

[68] Xiaohan Zhang, Haoqi Ye, Ziqi Huang, Xiao Ye, Yinzhi Cao, Yuan Zhang, and Min Yang. Understanding the (in)security of cross-side face verification systems in mobile apps: A system perspective. In *IEEE S&P*, 2023.

[69] Wei Zhou, Yan Jia, Yao Yao, Lipeng Zhu, Le Guan, Yuhang Mao, Peng Liu, and Yuqing Zhang. Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms. In *USENIX Security*, 2019.

[70] Xin'an Zhou, Jiale Guan, Luyi Xing, and Zhiyun Qian. Perils and mitigation of security risks of cooperation in mobile-as-a-gateway iot. In *CCS*, 2022.

[71] Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Automatic fingerprinting of vulnerable BLE iot devices with static uuids from mobile apps. In *CCS*, 2019.

[72] Chaoshun Zuo, Qingchuan Zhao, and Zhiqiang Lin. Authscope: Towards automatic discovery of vulnerable authorizations in online services. CCS, 2017.

Table 8: A full list of IoT-based device rental products studied in this work. (The mobility devices with gray backgrounds are e-scooters/e-bikes operated in Europe/US, and other devices with Chinese names are products only operated in China. Products with **bold name** have mobile apps, and the rest only provide WeChat mini-programs)

Device Type	Vendor
EV Chager	**Teld**(特来电) **Potevio**(普天新能源) **StarCharge**(星星充电) EVChargeNetwork(电动车充电网) Sunmue(尚e充电)
Charger	ShanKaiCharge(闪开来电) UUCharge(UU充电) MamCharge(猛犸充电) YunAn(世纪云安) NBLinks(涌鑫充电) EnergyMonster(怪兽充电） Dian(小电充电) DailyCharge(天天充电) BlueCatCharge(蓝猫共享充电) PisenPower(闪葱共享充电宝) QuQingTingCharge(趣蜻蜓共享充电) HeiQingTingCharge(黑蜻蜓充电) XiaoXunCharge(小巡共享充电) LuLuChong(路路充) TowerEnergy(铁塔充电) FlyPower(飞天鹰扫码充电器) QQCharging(千牛充电) DingDingCD(叮叮充电) Lvcc(驴充充) JieDian(街电) Xlvren(小绿人充电) BaJieCharge(八戒充电) DuDuBox(嘟嘟充电) BeiDian(倍电科技) ZhouDian(昼电共享)
Mobility	**Lyft Lime Bird Spin Dott Voi TIER Helbiz Neuron GO-ON Bolt**
	XiaoYuCX(小雨出行) KVCOOGO(快趣出行) XiaoMaCX(小玛电单) DiDiCX(滴滴出行) **HelloBike**(哈罗单车) **MeiTuanBike**(美团) FeiYueTu(飞跃兔出行) BossGo(Boss行) LeGeCX(乐哥出行) Liubike(小遛出行) LetFunGo(雷风出行) BaQiCX(巴骑出行) SunlightGo(阳光共享智行)MiMaDD(觅马出行QIQI(骑骑共享) XiaoHuangYaCX(小黄鸭共享) GongChi(共驰电单车) XiaoBeiCX(小呗畅行) XianLvGo(闲驴出行) **Hozonauto**(哪吒新能源) **DFPV**(东风新能源) QiXin(齐信共享单车) XiaoBinBike(小彬科技) ModaCX(摩达出行) YueHuoCX(月火出行) KeNaDianCX(克哪点出行) GXRongYi(共享荣熠惠行) YueShiJi(粤世纪共享)
Tools	Penguin Technology(企鹅共享) XiaoLianHB(智慧笑联) XiaoIZhiNengXiYi(小I智能洗衣) Smile-Iot(思迈尔智能) HQJL(华清捷利) Xiao-V(小v共享设备) AiPei(爱陪共享) WZ-Cloud(微众云) AnSheng(安圣科技) AnKong(安控)
Entertainment	AnMoJianKangGo(按摩健康GO) LeMoBar(乐摩吧) QSMX(骑思妙想) WeiPeng(微鹏) YFLe(易付乐) BDT(便电通)

A Real World Rentable Cellular IoT Products

Table 8 shows the IoT-based device rental products studied in this work. It falls into four usage categories: charging, mobility, entertainment, and daily tools.

- **Chargers.** With the proliferation of electric vehicles, rentable chargers are emerging in cities for people to charge everywhere. Some leading operators in China such as *Teld* [8], *StartCharge* [7], and *Potevio* [4] have deployed millions of rentable chargers. We also investigate many other rentable charging services, such as rentable power banks for mobile, smart sockets, and battery charging cabinets for electric bicycles.
- **E-Scooter/E-Bicycle/Car.** Compared to traditional rental services, e-scooters, bicycles, and cars have gained popularity due to their convenience. They can meet urban residents' flexible short-distance travel needs as massive unmanned devices are deployed everywhere, and people can rent them anytime in a pay-per-use mode via mobile apps.
- **Entertainment Devices.** Public entertainment devices such as massage chairs, billiards, and Mahjong tables can reduce management costs through IoT-based device rental services. With cellular IoT and companion apps, consumers can self-serve the process of paying, renting, and returning these devices.
- **Tools.** Facilities such as laundry machines and hair dryerss, are frequently used in our daily life. Using cellular IoT, these tools can be deployed and maintained unmanned in hotels or dormitories at low cost.

B Capturing Cellular Chip's Inner Traffic

As shown in Figure 8, we connect jumper wires to the IoT chips for power supply, message trace capture, and debugging. Some devices contains two chips and utilizing a cellular chip to relay messages from the control chip (i.e., main MCU). Their inter-chip traffic can be easily captured by using USB-TTL serial cables to monitor the messages between two chips.

For devices equipped with a single cellular chip, there are no inter-chip messages and the network traces come directly from the cellular chip itself. By enabling the debug log feature on these chips, we can access the unencrypted, raw messages from the cellular process log (CP log). We can efficiently capture these logs and retrieve the internal messages of the cellular chips by attaching jumper wires to the appropriate pinouts.

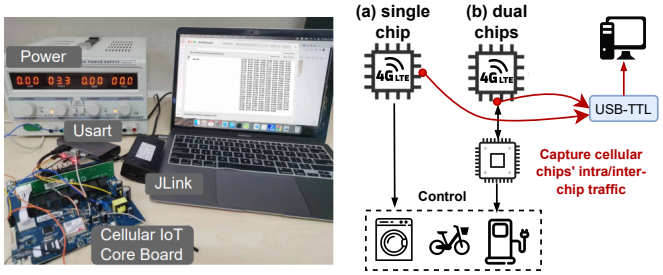


Figure 8: Capture cellular chips' inner traffic by connecting jumper wires to the logging pinouts.