E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem

Marco Casagrande
marco.casagrande@eurecom.fr
EURECOM
Sophia Antipolis, France

Riccardo Cestaro
riccardo.cestaro.1@studenti.unipd.it
University of Padova
Padova, Italy

Eleonora Losiouk
eleonora.losiouk@unipd.it
University of Padova
Padova, Italy

Mauro Conti
mauro.conti@unipd.it
University of Padova
Padova, Italy

Daniele Antonioli
daniele.antonioli@eurecom.fr
EURECOM
Sophia Antipolis, France

ABSTRACT

Xiaomi is the market leader in the electric scooter (e-scooter) segment, with millions of active users. It provides several e-scooter models and Mi Home, a mobile application for Android and iOS to manage and control an e-scooter. Mi Home and the e-scooter interact via Bluetooth Low Energy (BLE). No prior research evaluated the security of this communication channel, as it employs security protocols proprietary to Xiaomi. Exploiting these protocols results in severe security, privacy, and safety issues, e.g., an attacker could steal an e-scooter or prevent the owner from controlling it. In this work, we fill this research gap by performing the first security evaluation on all proprietary wireless protocols deployed to Xiaomi e-scooters from 2016 to 2021. We identify and reverse-engineer *four* of them, each having ad-hoc Pairing and Session phases. We develop *four* attacks exploiting these protocols at the architectural level, and we call them Malicious Pairing (MP) and Session Downgrade (SD). Both attacks can be performed from proximity, if the attacker's machine is within BLE range of the target e-scooter, or remotely, via a malicious application co-located with Mi Home. An adversary can utilize MP and SD to steal a password-protected and software-locked e-scooter, or to prevent a victim from accessing it via Mi Home. We isolate *six* attack root causes, including the lack of authentication while pairing, and the improper enforcement of the e-scooter password. We open-source the `E-Spoofer` toolkit. Our toolkit automates the MP and SD attacks, and includes a reverse-engineering module for future research. We empirically confirm the effectiveness of our attacks by exploiting three e-scooters (i.e., M365, Essential, and Mi 3), embedding five BLE subsystem boards and eight BLE firmware versions that support all four Xiaomi protocols. We design and evaluate *two* practical countermeasures that address our impactful attacks and their root causes, and we release them as part of `E-Spoofer`. We responsibly disclosed our findings to Xiaomi.

WiSec '23, May 29-June 1, 2023, Guildford, United Kingdom

ACM ISBN 978-1-4503-9859-6/23/05…$15.00
https://doi.org/10.1145/3558482.3590176

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**; *Hardware reverse engineering*.

KEYWORDS

Security, Xiaomi, Electric Scooter, Reverse Engineering

ACM Reference Format:
Marco Casagrande, Riccardo Cestaro, Eleonora Losiouk, Mauro Conti, and Daniele Antonioli. 2023. E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '23), May 29-June 1, 2023, Guildford, United Kingdom.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3558482.3590176

1 INTRODUCTION

Xiaomi is leading the electric scooter (e-scooter) market [20]. Its ecosystem includes seven e-scooters released in the last seven years (i.e., M365, Pro 1, Pro 2, 1S, Essential, Mi 3, and Mi 4) and the *Mi Home* mobile application for Android [3] and iOS [4]. Mi Home enables a user to manage his e-scooter, e.g., wirelessly locking and unlocking it or setting a password. Mi Home and the e-scooter communicate via *proprietary application-layer* protocols developed by Xiaomi. These protocols are undocumented, not peer-reviewed, and built on top of a *Bluetooth Low Energy (BLE)* link-layer.

Despite their associated security, privacy, and safety risks, no research work evaluated the security protocols used by Xiaomi to secure the interaction between its e-scooters and Mi Home. Instead, recent work focused on the privacy implications of e-scooter rental apps (including Xiaomi) [46] and on the security of Xiaomi's fitness tracking ecosystem [10]. In our work, we find that Xiaomi protocols can be exploited to (remotely) unlock and steal an e-scooter or permanently prevent its owner to manage it from Mi Home.

This work presents the *first* security evaluation of the communication channel between Xiaomi's e-scooters and Mi Home. In particular, we uncover and reverse-engineer all four e-scooter protocols used from 2016 to 2021. We label them as P1, P2, P3, and P4, and we dissect their custom Pairing (i.e., key agreement) and Session phases. We find that P1, P2, and P3 offer no security guarantees but *security through obscurity*. Instead, P4 provides some security properties (e.g., ECDH key agreement and AES-CCM authenticated encryption) but is vulnerable to *downgrade* attacks. Moreover, we

find that Xiaomi decided *not* to use standard BLE link-layer security mechanisms (e.g., BLE pairing), despite their devices support them.

We present *four* novel attacks targeting the Xiaomi protocols' specifications. Two attacks enable a proximity-based or remote attacker to pair maliciously with an e-scooter and get authorized access to it *without* spoofing the victim's identity (i.e., MP). The other two attacks allow a proximity-based or remote attacker to downgrade the connection with an e-scooter to an insecure version and send arbitrary commands (i.e., SD). The proximity-based adversary must be in BLE range of the target e-scooter. Instead, the remote adversary must have installed a malicious app on the victim's smartphone. Our attacks achieve *impactful* goals, such as unlocking and stealing an e-scooter, or preventing a victim from regaining control of the e-scooter via Mi Home. We isolate the *six* attacks' root causes, including the improper authentication and authorization mechanisms, and the unprotected but privileged vendor-specific features of Xiaomi protocols.

We release `E-Spoofer`, a toolkit capable of performing our four attacks by reimplementing and abusing the four reversed Xiaomi protocols. The toolkit includes *three* extensible modules. Two dedicated modules implement the Malicious Pairing and Session Downgrade attacks. The reverse-engineering (RE) module offers protocol dissectors to decode and build custom Xiaomi packets (e.g., P1, P2, P3, and P4). and useful Frida hooks for Mi Home to dynamically intercept and modify the proprietary Xiaomi payloads.

We successfully evaluate the attacks in *eight* different attack scenarios covering P1, P2, P3, and P4. Our setup allows testing multiple e-scooter configurations by using *three* modded e-scooters (e.g., M365, Essential, and Mi 3) with *five* BLE subsystems and *eight* BLE firmware. Our results are alarming. In all attack scenarios, we managed to unlock an e-scooter and steal it, or to lock it and to change its password, preventing its legitimate owner from accessing it via Mi Home. These results lead to millions [17] of exploitable devices.

To fix the four attacks and their six root causes, we developed and tested two usable and low-cost countermeasures and include them in our toolkit. First, we propose a backward-compatible pairing protocol with proper authentication and authorization mechanisms. Second, we provide a script to patch the session downgrade command from an e-scooter BLE firmware. We successfully test our patch on the M65 and Pro 1 e-scooters, whose BLE firmware is no longer updated by Xiaomi.

We summarize our contributions as follows:

- We present the first security evaluation of the proprietary security mechanisms employed by Xiaomi's e-scooters and Mi Home application. We isolate four custom application-layer security protocols on top of an insecure BLE link-layer. After reversing their Pairing and Session phases, we uncover six severe vulnerabilities in their design, including vendor-specific and unauthenticated protocol commands.
- We develop four attacks that steal an e-scooter or prevent its owner from accessing it from the Mi Home app previously paired with that e-scooter. The attacks are effective on P1, P2, P3, and P4, and can be deployed by an attacker in BLE range of a target e-scooter (i.e., proximity-based attacker) or via a malicious application on the victim's smartphone (i.e., remote attacker).
- We open-source `E-Spoofer`, an automated and low-cost toolkit that implements our attacks and tampers with the four Xiaomi protocols. Our toolkit includes the MP and SD attack modules, and a reverse-engineering module with protocol dissectors, firmware analysis tools, and Mi Home Frida hooks.
- We confirm that our four attacks are effective in eight attack scenarios covering five e-scooter BLE subsystems and eight BLE firmware. Our evaluation samples include P1, P2, P3, and P4. Our experimental setup allows to reproduce multiple attack scenarios using three partially disassembled e-scooters and different BLE subsystems. We also release two effective countermeasures that fix our attacks. The first addresses the MP attacks by implementing a more secure pairing protocol. The second prevents the SD attacks by patching the BLE firmware of an e-scooter.

Responsible disclosure and ethics. We responsibly disclosed our findings multiple times with Xiaomi via their bug bounty program [50]. In October 2022, we reported a UI password bypass issue with Mi Home, Xiaomi acknowledged it and provided a bug bounty. In November 2022, we shared a technical report and the code to reproduce our findings. In December 2022, we provided them with a video of the attacks on actual devices. Xiaomi did not follow up. We conducted our experiments in a controlled environment without involving third-party users and services. We anonymously provide our `E-Spoofer` toolkit at https://anonymous.4open.science/r/Espoofer-7B21 via a private repository that should stay confidential within the TPC. We will submit our toolkit for artifact evaluation. After responsible disclosure, we will open-source the toolkit.

2 XIAOMI E-SCOOTER ECOSYSTEM

Xiaomi is the electric scooter (e-scooter) market leader, sporting the highest number of active users and shipped devices [20]. Currently, it features seven e-scooters, i.e., M365 (2016), Pro 1 (2019), Pro 2 (2020), 1S (2020), Essential (2020), Mi 3 (2021), and Mi 4 (2022). Xiaomi also maintains *Mi Home*, a smartphone application for Android [3] and iOS [4] that manages Xiaomi's smart home devices, including any e-scooter. Xiaom's cloud-based backend service manages the e-scooters and their active Mi Home users.

Figure 1 shows a high-level representation of the Xiaomi e-scooter ecosystem. This work focuses on the BLE communication channel *between the e-scooter and Mi Home*. The e-scooter acts as a BLE peripheral (connection responder), while Mi Home is the BLE central (connection initiator). The e-scooter periodically broadcasts BLE advertisement packets to be discovered. These packets contain the e-scooter name, model, security level, and pairing mode activation. Mi Home scans the BLE spectrum and lists all connectable Xiaomi e-scooters nearby. Once connected, the devices exchange data using BLE's Generic Attribute Profile (GATT). The e-scooter exposes a GATT server, which includes the Nordic UART Service and a custom Xiaomi service. On the other hand, Mi Home acts as a GATT client, sending read, write, and subscribe requests to the e-scooter's GATT server. To communicate, Mi Home and the

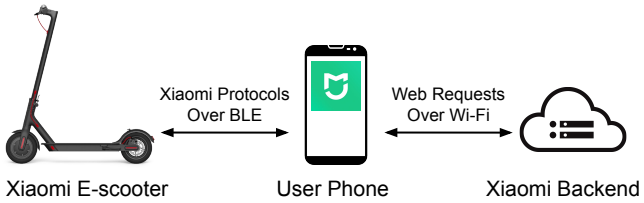

Figure 1: Xiaomi e-scooter ecosystem. Xiaomi e-scooter (left), the user smartphone running the Mi Home app (middle), and the Xiaomi backend (right). The e-scooter and the app are paired and connected over BLE. The app associates the e-scooter with the Xiaomi backend over Wi-Fi. We focus on the BLE traffic between the app and the e-scooter.

e-scooter establish a BLE link-layer connection. Then, they use *proprietary* application-layer protocols and mechanisms that cannot be scrutinized with multi-purpose static and dynamic analysis tools.

Mi Home requires the user to register a Xiaomi account to pair, connect, and manage one or more Xiaomi e-scooters. The pairing process is a one-time procedure that requires user interaction and an Internet connection. The user starts pairing via the app UI, scans for nearby Xiaomi e-scooters, selects the correct e-scooter from a list, presses the headlight button to activate pairing mode, and waits. Once the pairing is complete, Xiaomi backend links the user account to the paired e-scooter, and Mi Home remembers the device for future connections. Optionally, the user can set a 6-digit alphanumeric PIN to protect the e-scooter from unauthorized access to Mi Home (e.g., from attackers that have stolen the user's smartphone and want to unlock the e-scooter via Mi Home).

A Xiaomi e-scooter is a high-end embedded device composed of several *proprietary* and *undocumented* subsystems: *radio (BLE)*, *battery management (BMS)*, and *electric motor (DRV)*. Each subsystem has a dedicated system-on-chip (SoC) and firmware. The connection between the subsystems is not standardized and might involve a proprietary bus. The radio subsystem provides BLE connectivity, enabling communication between the e-scooter and Mi Home. It also acts as a gateway to distribute firmware updates to the DRV and BMS. The BMS monitors and manages the e-scooter's battery. The DRV takes care of the electric motor that, when the DRV is not up-to-date, can be patched to change the motor's maximum speed. At the time of writing, all Xiaomi e-scooters are manufactured by *Ninebot*, a Chinese company financed by Xiaomi that acquired Segway (its main competitor in the US) in 2015 [30].

3 THREAT MODEL

Now we present our system model and our proximity-based and remote attacker models. Please refer to Section 2 for their related background material.

3.1 System Model

We consider a victim who owns a Xiaomi e-scooter and a smartphone equipped with the Mi Home app for Android or iOS, as shown in Figure 1. We assume that the Mi Home version number is the *latest* available at the time of submission (e.g., Android v7.11.704 and iOS v7.12.204). We do *not* set a target Android or iOS version as we want to explore Xiaomi-compliant attacks that work regardless of the smartphone OS version.

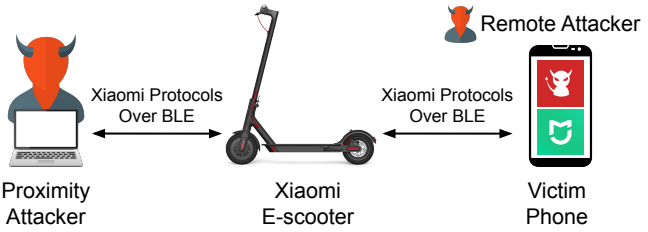

Figure 2: Proximity-based (left) and remote (right) attacker models investigated in this work. In the proximity-based threat model, the attacker is within BLE range of a target e-scooter. In the remote threat model, the adversary first installs a malicious Android app on the victim's smartphone. Then, she uses the malicious app (in red) to remotely target an e-scooter within BLE range of the victim's smartphone.

The victim securely paired the app, and the e-scooter accepted the required permissions and completed the default firmware update. The update process involves the BLE, battery management, and electric motor subsystems (e.g., DRV017, BLE157, BMS141), and the BLE component acts as a gateway. To consider the most secure scenario, we assume that the *password-protection* is enabled to prevent unauthorized access to the e-scooter. Hence, according to common sense, the victim locks and unlocks the e-scooter from the app. Moreover, the victim uses the e-scooter features, such as pressing the power button to activate or deactivate the headlight.

The e-scooter and Mi Home communicate using Xiaomi proprietary application-layer protocols. These protocols run on top of a link-layer connection established using BLE. *Only Xiaomi* knows the application-layer protocols' details and their security guarantees (e.g., confidentiality, integrity, and authenticity).

3.2 Attacker Models

Password protection and secure communication at the application-layer and link-layer should protect victims against impactful attacks, including threats effective from BLE proximity or remotely via a malicious app on the victim's smartphone. For example, it should not be possible to (remotely) unlock and steal an e-scooter or (remotely) reset a password to deny the victim access to the e-scooter. Based on this reasoning, and as shown in Figure 2, we focus on two relevant threat actors:

Proximity-based attacker. The proximity-based attacker targets the e-scooter with BLE signals. Hence, she requires being within BLE range of the target device. The proximity attacker has the following goals: (i) unlock and steal a (password-protected) e-scooter, and (ii) prevent the legitimate owner from accessing and controlling the e-scooter via Mi Home.

The proximity adversary has the capabilities of a real-world and low-cost BLE attacker. She can craft custom BLE packets, sniff the traffic over-the-air to get public information (e.g., BLE addresses and advertisements), and replicate the Android and iOS Mi Home apps with her attack equipment. The attacker does not observe the e-scooter while it pairs with Mi Home and does not install malicious

software on the victim's devices. Moreover, she does not physically tamper with the e-scooter and the smartphone (e.g., no physical fault injection and side-channel attack).

Remote attacker. The remote adversary attacks the e-scooter using a malicious application installed on the victim's smartphone. Thus, she requires the victim's smartphone to be within BLE range of the e-scooter, but she can remotely activate the app. For example, the adversary can attack the e-scooter while the victim is parking the e-scooter and walking away from the parking lot. This model differs from a proximity-based attack as the latter involves a BLE attacking machine (e.g., a laptop) in the BLE range of the victim, while the former involves a malicious smartphone app. The remote attacker has the same goals as the proximity-based attacker.

Capability-wise, we consider a low-cost and real-world remote threat actor targeting the *Android* ecosystem (as opposed to iOS, which is more closed). We assume a malicious Android app that was installed using known (yet practical) social engineering and phishing techniques. The app does not require root privileges but needs basic permissions to interact with the e-scooter, such as Bluetooth and Internet permissions. The attacker develops the app using standard Android tools (e.g., Android Studio) and APIs (e.g., BLE advertisement, scanning, and GATT APIs). The remote attacker has the same limitations as the proximity one, except for installing an app on the victim's smartphone.

Physical access requirements. Regardless of the attacker model, we assume that the adversary needs minimal (but mandatory) physical access to steal and carry away an e-scooter. For example, in a proximity-based scenario, the attacker can approach the e-scooter when the victim is not present and perform some short interactions with its dashboard (e.g., pressing the headlight and the power buttons). Alternatively, in a remote threat scenario, two adversaries can collude. For example, an adversary unlocks the e-scooter by launching a remote attack via the malicious app. At the same time, the other adversary can press any button (if necessary) and steal the e-scooter.

4 REVERSED XIAOMI SECURITY PROTOCOLS

We describe the *four* proprietary Xiaomi protocols that we reverse-engineered (RE). Please see the Appendix for an explanation of our RE methodology. We discover that Mi Home and the e-scooter establish an *insecure* link-layer BLE connection, despite both devices supporting BLE security mechanisms (e.g., BLE Pairing). Instead, Xiaomi uses *proprietary application-layer protocols* to secure their whole e-scooter ecosystem.

Table 1 summarizes the details we reversed from the protocols. We label the protocols as P1, P2, P3, and P4, and also assign a descriptive name to each one. P1 is named "No security" because it does not utilize any security mechanism. P2 is named "XOR obfuscation" because it employs an obfuscation strategy exclusively based on XOR. P3 is named "AES-ECB and XOR obfuscation" because it XORs Xiaomi packets with the output of an AES-ECB cipher. P4 is named "ECDH and AES-CCM" because it employs ECDH for Pairing and AES-CCM during Session. Then, we isolate the protocols' phases: *Pairing* (e.g., key agreement), and *Session* (e.g., authenticated encryption). For instance, P2 Pairing is based on a public XOR mask and is unauthenticated. Its Session reuses the XOR mask to obfuscate payloads, is not authenticated, and provides no integrity protection. Our experiments reveal that all Xiaomi e-scooters (more specifically, their BLE subsystems) and all Mi Home versions (from 2016 to 2021) have employed these protocols. Now we describe each protocol in detail.

4.1 No Security (P1)

P1 provides no security guarantees as it lacks Pairing and Session capabilities. The devices establish a BLE connection and then exchange the application-layer payloads in cleartext without integrity protection. The only roadblock for the attacker to eavesdrop and inject packets into the connection is the knowledge of the application-layer packet format. P1 is the prototypical example of *security through obscurity (STO).*

4.2 XOR Obfuscation (P2)

P2 offers no security guarantees, but relies on a XOR-based obfuscation strategy. During Pairing, Mi Home reads a twelve-byte *XOR mask* from the e-scooter Hardcopy Data Channel GATT characteristic, different at every reboot of the device. Then, during Session, the devices obfuscate the application-layer payloads by XORing them with the XOR mask. If the payload is longer than the XOR mask, the app asks the e-scooter for the extended version of the same XOR mask and uses that one instead in the XOR operation. Since the attacker can trivially recover the mask (e.g., eavesdropping or reading it from the e-scooter), P2 is insecure and falls into the STO category.

4.3 AES-ECB and XOR Obfuscation (P3)

P3 uses a weak key establishment protocol based on AES-ECB and XOR obfuscation. Pairing generates a sixteen-byte pairing key (`pk`) by computing `pk=AES-ECB(key=constant,input=escooter_name)`, where `constant` is *hardcoded* both in the Mi Home app and in the e-scooter BLE firmware, and `escooter_name` is publicly advertised by the device. Then, during Session, the devices obfuscate the application-layer payloads by XORing them with `pk`. If the payload is longer than the pairing key, the payload is XORed with an extended pairing key, which is just `pk` repeated as many times as necessary. P3 provides no security guarantees but only STO. An attacker can compute `pk` by extracting `constant` from the reversed code of any Mi Home APK and trivially acquire `escooter_name`. Once `pk` is known, the attacker can de-obfuscate and inject valid P3 packets.

4.4 ECDH and AES-CCM (P4)

During Pairing, P4 employs *Elliptic Curve Diffie-Hellman (ECDH)* for key agreement and unilateral pairing key authentication. In particular, the e-scooter sends `chal`, a sixteen-byte random challenge. The devices exchange their public keys, using the *SECP256R1* curve, and derive `ss`, an ECDH shared secret. Then, they compute a pairing key (`pk`) and a one-time key (`otk`) using HKDF as follows: `pk||otk=HKDF(key=ss,input="mible-setup-info",salt="")`. The app responds to the e-scooter challenge with `resp=AES-CCM(key=otk,input=chal)`.

Table 1: The four Xiaomi application-layer security protocols analyzed in this work. The first and second columns show the protocol ID and name. Each protocol has a Pairing and Session phase. P4 has two Session versions, where v2 is equal to v1 but adds downgrade protection. Unil means unilateral.

ID	Name	Pairing	Session
P1	No security	None	None
P2	XOR obfuscation	Public XOR mask, no auth	XOR mask obfuscation, no auth, no integrity
P3	AES-ECB and XOR obfuscation	Weak AES-ECB key agreement, no auth	XOR obfuscation, implicit auth, no integrity
P4	ECDH and AES-CCM	ECDH, AES-CCM unil auth	v1: HKDF, HMAC, AES-CCM, mutual auth v2: v1 with downgrade protection

During Session, P4 uses HKDF, to derive the directional session keys, and HMAC-based mutual authentication. The devices exchange `rand_esc` and `rand_app`, two sixteen-byte random numbers. The devices derive two directional session keys (`sk_esc` and `sk_app`) and AES-CCM nonces (`n_esc` and `n_app`) as follows: `sk_esc||sk_app||n_esc||n_app=HKDF(key=pk,input="mible-login-info",salt=rand_app||rand_esc)`.

Then, the e-scooter sends `resp_esc=HMAC(key=sk_esc,input=rand_esc||rand_app)` to authenticate its session key. Similarly, the app authenticates its directional key by sending `resp_app=HMAC(key=sk_app,input=rand_esc||rand_app)`. After mutual authentication of both session keys, each device employs AES-CCM to encrypt and integrity protect the application-layer payloads. AES-CCM is keyed with the directional session key and initialized with the directional nonce concatenated with a packet counter.

P4 provides security guarantees (unlike P1, P2, and P3) but can be downgraded. Replay attacks are ineffective against P4 Pairing and Session because the former utilizes a random challenge during pairing key authentication, and the latter utilizes random values and nonces during the HMAC-based authentication. Moreover, the mutual authentication during P4 Session prevents impersonation attacks on Mi Home or the e-scooter. The usage of session keys limits the impact of a compromised key to the current session only, and the usage of a packet counter in the encryption of regular BLE communication protects against nonce reuse attacks.

P4 protocol comes in two versions (i.e., P4v1 and P4v2), depending on the supported version of the Session phase.

5 ATTACKS

We present *four novel attacks* targeting the four Xiaomi custom protocols discussed in Section 4 that enable stealing a (password-protected) e-scooter or denying a victim from using it via Mi Home. Our attacker can either be *proximity-based* or *remote*, as stated in Section 3. The attacks achieve their goals by using one of two spoofing strategies: (i) the attacker pairs with the target e-scooter while impersonating any user, i.e., *Malicious Pairing (MP)* (ii) the adversary connects to the target e-scooter and downgrades the session to an insecure version, i.e., *Session Downgrade (SD)*.

The attacks are critical to the Xiaomi ecosystem as they exploit the four Xiaomi application-layer security protocols at the *architectural level*. Hence, they are effective regardless of the e-scooter's hardware and software details, including its model, and only depend on the BLE firmware being run. Moreover, they defeat the most secure setup, i.e., a password-protected and software-locked e-scooter already paired with a registered Xiaomi user. We even completed the attacks while the e-scooter was in motion (in a controlled environment). We now describe the MP and SD strategies, and we isolate their root causes.

5.1 Malicious Pairing (MP)

Figure 3 shows the MP attack strategy that can be used to lock an e-scooter away from its user, or to steal it. The attacker waits until the victim presses the e-scooter headlight button to switch on or off the front light (or presses the button if the e-scooter is unattended). As a *side effect*, the button press activates pairing mode for the e-scooter for *seventeen seconds* without notifying the user. The adversary

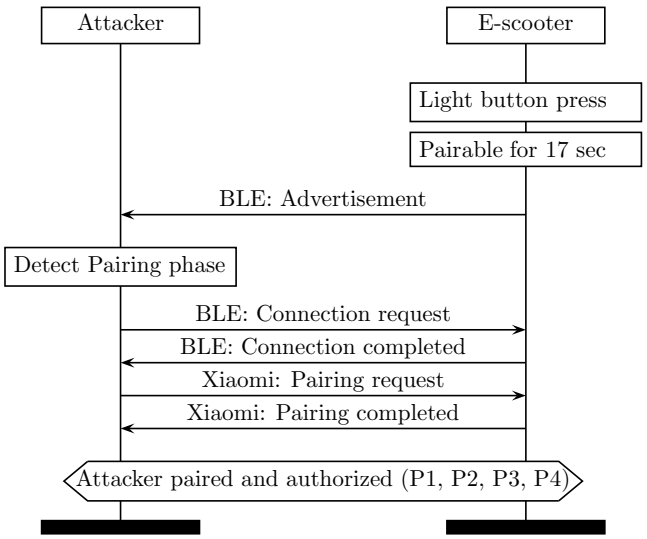

Figure 3: Malicious Pairing (MP) attack strategy. The user presses the headlight button. The e-scooter goes into pairable mode for seventeen seconds and advertises it via BLE. The attacker detects the Pairing phase supported by the e-scooter. Then, she establishes a BLE connection without impersonating the victim's smartphone and completes Xiaomi Pairing. As a final result, she is authorized to send any Xiaomi-compliant command to the e-scooter, including lock, unlock, and set or change a password.

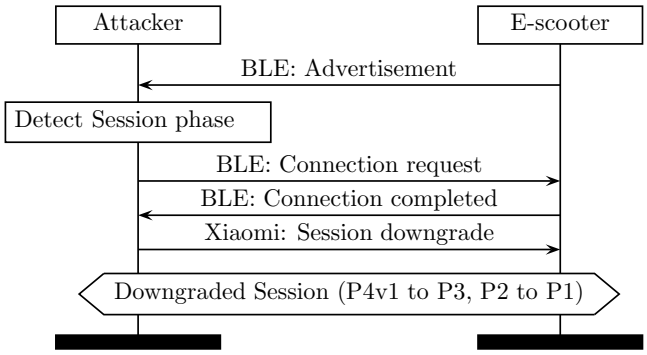

Figure 4: Session Downgrade (SD) attack strategy. The app detects a nearby e-scooter vulnerable to SD (i.e., running P4v1 or P2). The attacker skips Pairing and sends the session downgrade command to the e-scooter. The Session is downgraded from P4v1 to P3, or from P2 to P1.

detects that the e-scooter is pairable from its BLE advertisement packets and detects which Xiaomi protocol it supports (i.e., P1, P2, P3, or P4). Then, she establishes a BLE link-layer connection *without* spoofing the victim's smartphone BLE address. Hence, the adversary can target an e-scooter without knowing any information about its owner (e.g., any e-scooter in a parking lot).

Finally, the attacker sends a Xiaomi-compliant pairing request and completes Pairing, regardless of the supported Xiaomi protocol of the e-scooter. Once paired, she can perform any action requiring authentication. For example, she can lock it and set a new e-scooter password to prevent the victim from accessing it from Mi Home. The takeover is effective, as we discovered that Mi Home does *not* allow resetting the e-scooter password, even with a factory reset. Alternatively, the attacker can use the MP strategy to unlock and steal the e-scooter.

The MP attack strategy is effective for a proximity-based attacker inside the BLE range of the e-scooter, and for a remote attacker controlling a malicious app while the victim's smartphone is within BLE range of the e-scooter. Moreover, the strategy works regardless of the Pairing phase version and the e-scooter password because, while pairing, the attacker does not have to authenticate its identity and provide the password.

5.2 Session Downgrade (SD)

Figure 4 shows the SD attack strategy that allows an adversary to lock an e-scooter away from its user, or to steal it. The attacker detects the Session protocol supported by the e-scooter. Then, she looks at the BLE advertisement packets of the e-scooter, and she detects if the target runs P4v1 or P2, being the two Session protocols that expose a session downgrade command. She establishes a BLE link-layer connection without spoofing anything from the victim. Hence, the adversary can target an e-scooter running P4v1 or P2 without knowing any information about its owner. Then, the attacker sends a Xiaomi-compliant session downgrade command, downgrading the Session from P4v1 to P3 or from P2 to P1. The attacker exploits the insecure P3 and P1 to perform dangerous actions on the e-scooter. Similarly to MP, she can lock the e-scooter and prevent access to it from Mi Home by setting a new e-scooter password. She can also use the SD strategy to unlock and steal the e-scooter. The SD strategy can be applied to our proximity-based and remote threat models. The strategy entirely skips Pairing and starts an insecure downgraded Session, removing any authentication requirement from the attacker. Moreover, the strategy is particularly effective on e-scooters running P4v1, as they offer security guarantees that are nullified by downgrading the Session phase to the insecure P3.

5.3 Root Causes

The four attacks presented above are enabled by the following *six* root causes (i.e., vulnerabilities) that we isolated in the Xiaomi protocols' specification:

V1: Unauthenticated Pairing. None of the Pairing phases require *device authentication* (e.g., via a certificate signed by Xiaomi). Hence, an attacker can pair with an e-scooter while spoofing an arbitrary Mi Home app without authenticating, regardless of the application-layer protocol used by the e-scooter (i.e., P1, P2, P3, or P4).

V2: Unintentional Pairing mode. Pressing the e-scooter's headlight button activates pairing mode for *seventeen seconds* without notifying the user. Hence, whenever the victim presses the headlight button, an attacker in the BLE range of the e-scooter can detect that the e-scooter is pairable from its BLE advertisements and pair. Alternatively, given physical access, the attacker can trigger pairing mode while the victim is away, by simply pressing the headlight button (even if the e-scooter is software-locked).

V3: Improper e-scooter password enforcement. The e-scooter does *not* enforce the password set by the user via Mi Home. Only Mi Home checks it to prevent unauthorized access to the e-scooter from the victim's smartphone. Therefore, an attacker can tamper with a password-protected e-scooter without knowing the password. Moreover, Mi Home does not provide a way to *deactivate* the password, and the password does *not* change across factory resets. If the adversary changes the e-scooter password, she prevents the victim from controlling the e-scooter via Mi Home.

V4: Unprotected sensitive memory. Xiaomi custom protocols include an unauthenticated command to read and write *sensitive* memory regions. For example, the attacker can read and overwrite the victim's password from the e-scooter DRV subsystem memory. Moreover, she can tamper with the BLE subsystem memory to lock, unlock, reboot, and shut down the e-scooter.

V5: Downgradable and insecure Session. Xiaomi custom protocols include unauthenticated commands to downgrade the Session phase. For instance, the attacker can downgrade a P4v1 Session to a P3 Session and a P2 Session to a P1 Session. At the same time, P1, P2, and P3 Session phases are insecure and provide no confidentiality, authenticity, or integrity guarantees. P1 uses no key, P2 employs XOR-based obfuscation with a constant XOR mask, and P3 uses a slightly more complex, yet predictable, obfuscation based on AES-ECB and XOR operations.

V6: No BLE security despite device support. Xiaomi does not employ BLE security at the link-layer despite device support but relies solely on its custom security mechanisms at the application-layer.

So, there is no defense in depth, and the application-layer is a single point of failure.

Table 3 in the Appendix maps the six root causes of the four attacks described earlier. MP attacks exploit V1, V2, V3, and V4. V1 and V3 lower the attack requirements. V2 allows attacks from proximity without requiring physical access to activate pairing mode. V4 enables dangerous operations on the e-scooter by unauthorized attackers. SD attacks exploit V3, V4, V5, and V6. V3 and V6 lower the attack requirements. V4 enables dangerous operations on the e-scooter. V5 makes SD possible.

6 IMPLEMENTATION

Here, we present `E-Spoofer`, a new toolkit to carry out the four attacks presented in Section 5, facilitate further reverse-engineering of Xiaomi protocols and help in future security evaluations in the Xiaomi e-scooter ecosystem.

6.1 Proximity Attack Module

The `E-Spoofer` proximity attack module performs proximity-based MP and SD *over-the-air*, using BLE. We use Noble [23], a NodeJS module, to create a BLE central that spoofs the Mi Home app and speaks Xiaomi protocols. We replicate P4 Pairing and P4v1 Session, as these protocols are available on all (up-to-date) Xiaomi e-scooters.

Our module reimplements P4 Pairing, including ECDH and pairing key authentication. We perform ECDH and obtain a shared secret. We receive a challenge from the e-scooter. We derive a pairing key and a one-time key from the shared secret by running HKDF-SHA256. We utilize the one-time key and the challenge for the sophisticated pairing key authentication by running AES-CCM-128. Finally, we send the solution to the e-scooter and complete P4 Pairing.

Our module reimplements P4v1 Session, including the HMAC-based mutual authentication and AES-CCM encryption. We send a challenge to the e-scooter, and receive a challenge from him. We retrieve the pairing key generated during Pairing. We derive the directional session keys and IVs from the pairing key and the two challenges by running HKDF-SHA256. Then, we use the directional session keys and IVs to calculate the solution of the e-scooter challenge by running HMAC-SHA256. Finally, we encrypt with AES-CCM-128 the BLE commands (e.g., session downgrade, lock or unlock the e-scooter, setting or changing the password) using the session keys and IVs, and the packet count. The above-mentioned cryptographic operations also require other input values found in the decompiled Mi Home code, identical for all e-scooter models.

6.2 Remote Attack Module

The `E-Spoofer` remote attack module performs the attacks using a *malicious Android app*. The app acts as a BLE central, spoofing Mi Home and speaking Xiaomi protocols. It detects a vulnerable e-scooter via its BLE advertisement, by analyzing the info included in the advertisement itself (i.e., e-scooter name, model, security level, and pairing mode activation). When an e-scooter is found in pairing mode, the app will pair and perform MP or SD. We develop the app using the RxAndroidBle library [40], built on RxJava.

Our malicious app requires no root privileges but Bluetooth and location-related permissions. On Android 9 or lower, these permissions are BLUETOOTH, BLUETOOTH_ADMIN, and ACCESS_COARSE_LOCATION. Android 10 and 11 require ACCESS_FINE_LOCATION instead of coarse locatio. On Android 12 or higher, the app requires the BLUETOOTH_CONNECT and BLUETOOTH_SCAN permissions.

6.3 Reverse-Engineering Module

The `E-Spoofer` reverse-engineering module contains the protocol dissectors, Ghidra utilities, and Frida hooks that we developed while statically and dynamically RE the Xiaomi e-scooter ecosystem. The research community can use these modules to perform other experiments on the Xiaomi ecosystem or adapt them to test similar ecosystems. We now describe each submodule.

Protocol Dissectors. We develop Pyshark *dissectors* that automatically parse BLE captures and detect custom Xiaomi payloads and advertisement packets. They identify the Xiaomi protocol version from the packet header and dissect the packet accordingly. We also develop Scapy scripts to complement the Pyshark dissectors and offer a more advanced analysis.

We develop an advertisement packet analyzer for Xiaomi e-scooters. Our script extracts the name of the scooter (e.g., MIScooter1234), the scooter model (i.e., 0x20 for M365), the security level (i.e., 0x00 for P1, 0x01 for P2, and 0x02 for P3 and P4) and pairing mode activation (i.e., 0x01 means not active, 0x02 means active).

Ghidra Utilities. We utilize *Ghidra* [27] to statically RE portions of the e-scooter's BLE firmware. We used the open-source mijia library [22] to identify some compiled functions in the firmware, related to BLE advertisement and cryptographic mechanisms (e.g., AES, HKDF). We manually name the functions related to Pairing and Session and release six YARA [47] rules with their signatures to identify them automatically. We also release our Ghidra project files to reproduce our setup, as part of `E-Spoofer`.

We discover how the session downgrade command is implemented in the BLE firmware, and why P4v2 does not support it. A static memory *flag* decides whether P3 packets (including the downgrade command) are accepted or discarded. Firmware running P4v1 enables this flag, thus becoming vulnerable to SD. Firmware running P4v2 disables this flag, thus discarding the downgrade command and becoming immune to SD. We did not find any way to exploit this flag, unreachable by the *unprotected sensitive memory* (V4) root cause presented in Section 5.3.

Frida Hooks. First, we decompile the Mi Home APK. We navigate the decompiled code to find the classes and functions involved in Xiaomi security mechanisms and we write down their signature. Then, we develop Frida [29] hooks to intercept these calls. We print the input and output values, and we modify them if needed. In particular, we cast the key to their proper classes, before printing or altering them. Our hooks are written in Javascript and can be run by invoking the Frida client from the console, while connected to a rooted smartphone running a Frida server. Operating with Mi Home, will print logs on the console.

7 EVALUATION

In this section, we evaluate the four attacks presented in Section 5 against *eight* attack scenarios. We cover P1, P2, P3, and P4 – the proprietary Xiaomi application-layer protocols reversed in Section 4, three popular Xiaomi e-scooters models (i.e., M365, Essential, and Mi 3), five Xiaomi BLE subsystems (i.e., M365, Pro 1, Pro 2, Essential, and Mi 3), eight e-scooters' BLE firmware, and the Mi Home app for Android and iOS. We now describe our setup and results.

7.1 Setup

Our evaluation setup enables experimenting with multiple e-scooters and BLE configurations by using three e-scooters (M365, Essential, and Mi 3) configured to host different BLE subsystems and firmware. We bought the three e-scooters from Amazon for around 1.000 USD. We get access to their BLE subsystem board by unscrewing the dashboard and removing the display. This way, we broaden our evaluation while limiting the evaluation costs. For example, by installing the Pro 1 and Pro 2 BLE subsystems and firmware on the M365 e-scooter, we can test the Pro 1 and Pro 2 subsystems *without* spending hundreds of USD to buy the actual e-scooter.

We test five BLE subsystem boards with eight BLE firmware. Three boards are original parts of M365, Essential, and Mi 3 e-scooters. Two are clone boards for Pro 1 and Pro 2. The M365 and Pro 1 subsystems include an *nRF51822* SoC [39] of the QFAA variant (16 KB of RAM). Instead, the other subsystems use the QFAC variant with 32 KB of RAM. We obtain the BLE firmware from the ScooterHacking repositories [34] or the Mi Home app. We identify each firmware's relative proprietary protocol (i.e., BLE072 runs P1, BLE081 runs P2, BLE090 runs P3, BLE122, BLE129, BLE152, and BLE153 run P4v1, BLE157 runs P4v2).

To debug and manage the BLE subsystems, we use the *ST-Link V2 debugger* [14], which is compatible with the nRF51 SoC family. Attaching the debugger to a subsystem board requires manual effort, such as soldering the data (SWDIO), clock (SWCLK), and power wires. We also remove discrete components to unlock hardware-based debugging (i.e., C16 and R1 on the M365 BLE board, C2 on the Pro 1 BLE board). Once debugging was unlocked, we could run a GDB server for runtime debugging and operate on the SoC RAM with tools such as OPENOCD [28], PySWD [31], MiDu Flasher [18], and nRFSec [7]. Runtime access to the subsystem boards was essential to produce the presented results. For example, via GDB, we discovered that the e-scooters store the cleartext password in RAM, and via firmware flashing, we restored a BLE subsystem in an unbricked state after tampering with it.

On the app side, we test Mi Home for Android and iOS on three smartphones. We evaluate a rooted Pixel 2 running Mi Home v7.11.704 and Android 11, a rooted Oneplus 3 with Android 9 and a Realme GT with Android 12, both running Mi Home v7.6.704, and an iPhone 7 running Mi Home 7.12.204 and iOS v15.7. Our attacks do *not* require rooting a smartphone; we only need root privileges when dynamically instrumenting Mi Home with Frida.

We run `E-Spoofer`, the novel toolkit we present in Section 6, from two attacking devices. We deploy our proximity-based MP and SD attacks from a laptop (i.e., Dell Inspiron 15 3000). We select the desired attack from the command line, the victim e-scooter from a list of nearby targets, and the script automatically performs MP or SD, displaying visual feedback. We deploy our remote MP and SD attacks from a smartphone (e.g., Pixel 2). Through the UI of our malicious app, we scan for nearby targets, connect to a victim e-scooter, and perform MP or SD.

7.2 Results

Table 2 shows our evaluation results. The first two columns indicate the BLE firmware version and the protocol they run. The third column represents the e-scooter model, which hosts the BLE subsystem shown in the fourth column. We specify the SoC variant of the BLE subsystem board in column five. The remaining columns highlight whether a BLE firmware version is vulnerable to MP and SD in their proximity-based and remote variant.

In our attack scenarios, we exploit eight unique BLE firmware, including the latest firmware available on the M365, Essential, and Mi 3. We test the four Xiaomi proprietary protocols we identified, including the two variants of P4 Session (i.e., P4v1 and P4v2), and flash them on five BLE subsystems from different e-scooter models. We confirm that BLE subsystems using the nRF51822 QFAA SoC are incompatible with newer e-scooters models (i.e., Essential, Mi 3), as the latter requires BLE subsystem boards with the nRF51822 QFAC SoC. Similarly, newer boards cannot be installed on the M365. We demonstrate that *all* evaluated BLE subsystems, regardless of their application-layer protocol, are vulnerable to the MP attacks. This happens due to authorization and authentication issues in all four Xiaomi protocols that we discuss and fix in Section 8. We also demonstrate that *all* evaluated BLE subsystems running P4v1 or P2 are vulnerable to SD to P3 or P1. We highlight that P1, P2, and P3 have no security guarantees compared to the more secure P4. This fact makes SD from P4v1 to P3 particularly threatening. We confirm that P4v2 is immune from the SD attacks, as discussed in Section 6.

Our `E-Spoofer` toolkit proved to be effective on all evaluated Xiaomi e-scooters. Unfortunately, we could not evaluate the Xiaomi Mi 4 e-scooter due to its release time (end of 2022). `E-Spoofer` can be easily extended to support any e-scooter ecosystem that protects their communications with a proprietary application-layer protocol on top of BLE, including the Xiaomi Mi 4 e-scooter. To attain this goal, future researchers will have to reverse-engineer the proprietary application-layer protocols run by that specific e-scooter ecosystem. In the Appendix, we present our reverse-engineering methodology, which is generalizable to any BLE e-scooter and utilizes state-of-the-art tools and techniques. We also confirm that our toolkit can change the unknown e-scooter password set by an adversary, restoring the user capability of accessing and managing the e-scooter from Mi Home, as a post-attack defence.

During our experiments, we even identified and disclosed a severe *UI authentication* bug in Mi Home for Android and iOS. From Mi Home v7.6.704 onwards, the user can lock or unlock a password-protected e-scooter *without* entering the password. The cause is a 1 second UI delay between the app wake-up and the password prompt. We confirmed this bug using the same smartphones we describe in Section 7.1. Since the password is only checked by Mi Home, due to the `improper e-scooter password enforcement` (V3) root cause we discuss in Section 5.3, the attacker can bypass app-based password protection, unlock the e-scooter, and steal it.

As described in the responsible disclosure paragraph, Xiaomi acknowledged this bug, rewarding us with a bounty, but gave no information about a fix.

8 COUNTERMEASURES

To address the four impactful attacks described in Section 5, we design and evaluate two *usable*, *backward-compliant*, and *low-cost* countermeasures. The first countermeasure stops the MP attacks by providing a stronger pairing mechanism that is appropriately authorized and authenticated. The second countermeasure fixes the SD attacks by patching away the hidden downgrade command from the vulnerable e-scooter BLE firmware. We now describe them in detail and release them as part of `E-Spoofer`.

8.1 Authorized and Authenticated Pairing

The MP attacks presented in Section 5.1 are enabled by authorization and authentication issues affecting P1, P2, P3, and P4 Pairing phases. We develop a better pairing phase addressing both issues in a backward-compatible way. This countermeasure addresses the *unauthenticated pairing* (V1), *unintentional pairing mode* (V2), and *improper e-scooter password enforcement* (V3) root causes from Section 5.3. We now describe how we provide authorization and authentication during pairing.

Authorized Pairing Mode. We require the Xiaomi Pairing phase to implement a *dedicated* pairing activation command that also *notifies* the user. In particular, to enter pairing mode, the user must press the headlight button while holding down the left brake. Then, the e-scooter's tail light should blink until the completion of Pairing. This fix prevents unexpected and unnotified pairing sessions such as the ones exploited in the MP attacks by waiting until the victim presses the headlight button. The fix is trivial to implement for Xiaomi as it requires minimal modifications to the BLE firmware. On our side is challenging to test as we do not have access to the BLE firmware source code and build tools.

Password-Protected Authenticated Pairing. We require a password protected pairing protocol to prevent an unauthenticated attacker from pairing with a victim e-scooter. This fix prevents the MP attacks even if the adversary manages to put the e-scooter in pairing mode. This countermeasure is easy to implement by extending the Mi Home password protection functionality. In particular, while pairing an e-scooter with Mi Home for the first time (including after a factory reset), the user should set a password via Mi Home. Then, the password should be stored on Mi Home and the e-scooter and enforced in case of re-pairing. Hence, an attacker cannot maliciously pair with the e-scooter as she cannot provide the password to the e-scooter. We successfully evaluated this fix using our toolkit to replicate P4 Pairing between an e-scooter and Mi Home.

8.2 Anti-Downgrade BLE Firmware Patching

The SD attacks presented in Section 5.2 are enabled by a vendor-specific command, which downgrades Xiaomi Session P4v1 to P3, and P2 to P1. We focus on patching P4v1 because e-scooter running the insecure P2 should update their BLE firmware to the latest version. Regardless, the downgrade command is present even in recent BLE firmware versions, including the latest M365 and Pro 1 BLE firmware. We release a script capable of finding and removing the downgrade command from a vulnerable BLE firmware to fix this issue. Our script addresses the *downgradable and insecure Session* (V5) root cause presented in Section 5.3.

The script looks for a specific conditional statement and patches it to allow only P4 Session. Hence, the patch introduces no overheads (e.g., memory, computation). Our scripts opens the binary firmware, finds the function responsible for BLE packet analysis, and alters the conditional statement that accepts either P3 and P4 packets, causing it to only accept P4 packets. More specifically, it replaces the `cmp` instruction `5a2f` with `552f`. As a result, the attacker can neither downgrade P4v1 to P3, nor send any other insecure P3 command.

Developing the script required a one-time manual overhead to understand how to remove the downgrade command. Then we *automated* our binary-patching process. We reuse the BotoX M365 patcher tool [6] to encrypt the patched firmware with the Tiny Encryption Algorithm (TEA). We reuse the third-party M365DownG app [9] to flash the zipped and newly encrypted BLE firmware.

We successfully evaluated our fix on the M365 and Pro 1 e-scooters. We flashed a patched BLE122 firmware on the e-scooters and deployed the proximity-based and remote SD attacks. Both attacks failed, as downgrading the protocol from P4v1 to P3 was impossible with our fix.

9 RELATED WORK

E-Scooters Security and Privacy Issues. Academic research on e-scooter security and privacy is scarce, especially on personal e-scooters. Zimperium, a mobile security company, exploits the locking system to stop a running e-scooter [52]. The hacker Lanrat evaluated M365 authentication, discovering that it is not enforced by the e-scooter [16]. Both attacks were publicly disclosed in 2019 and only targeted the Xiaomi M365 model. In our work, we target all Xiaomi e-scooter models from 2016 to 2021.

Security researchers focused on e-scooter rental ecosystems instead of private e-scooters. In [1], the authors identify some vulnerabilities in the APIs exposed by the Bird e-scooter sharing platform, which utilizes M356 e-scooters [5]. Public e-scooters from the Lime sharing company are weak to a man-in-the-middle attack that allows for arbitrarily swapping audio files [26]. N. Vinayaga-Sureshkanth et al. [46] provide an extended evaluation of Android e-scooter rental applications. In particular, they investigate the user-related data collected and shared with third parties, which could monitor the users' schedules and visited locations. In our work, we perform a security assessment. Therefore, we consider out-of-scope any privacy study on user data.

E-Scooters Hacking Communities. ScooterHacking [37] is the largest e-scooter hacking community with around 20.000 members. ScooterHacking releases hacking tools [34] and offers a third-party companion app [36] for Xiaomi e-scooters. Expert users can download custom DRV and BLE firmware to alter the e-scooter performances (e.g., maximum speed). Alternatively, users can build their DRV firmware with the ScooterHacking Custom Firmware Toolkit [33] and the BotoX Xiaomi M365 Firmware Patcher [6]. These tools offer limited customizability as they can only binary patch hardcoded and unsigned portions of the firmware.

Table 2: Evaluation results. The first and second columns represent the BLE firmware version and the Xiaomi protocol version. The third column states the e-scooter model, which hosts the BLE subsystem board, specified in the fourth column, indicating if the BLE board is original from Xiaomi or a clone. The fifth column specifies the System-on-Chip present on the BLE subsystem. The last four columns highlight if the evaluated combination is vulnerable to our proximity-based and remote Malicious Pairing (MP) and Session Downgrade (SD) attacks. A hyphen (-) means the attack does not apply to that target.

					Proximity		Remote	
Firmware	**Protocol**	**E-Scooter**	**BLE Sub. Board**	**SoC**	**MP**	**SD**	**MP**	**SD**
BLE072	P1	M365	M365 (Original)	nRF51822 QFAA	✓	-	✓	-
BLE081	P2	M365	M365 (Original)	nRF51822 QFAA	✓	✓	✓	✓
BLE090	P3	M365	Pro 1 (Clone)	nRF51822 QFAA	✓	✗	✓	✗
BLE122	P4v1	M365	M365 (Original)	nRF51822 QFAA	✓	✓	✓	✓
BLE129	P4v1	M365	Pro 2 (Clone)	nRF51822 QFAC	✓	✓	✓	✓
BLE152	P4v1	Essential	Essential (Original)	nRF51822 QFAC	✓	✓	✓	✓
BLE153	P4v1	Mi 3	Mi 3 (Original)	nRF51822 QFAC	✓	✓	✓	✓
BLE157	P4v2	Mi 3	Mi 3 (Original)	nRF51822 QFAC	✓	✗	✓	✗

Third-party researchers provided non-peer-reviewed blog posts about the BLE traffic exchanged by some Xiaomi e-scooters [8, 12, 24, 35]. These resources helped in the initial stage of our work but failed short on the technical details and e-scooter coverage. For example, some report confuses encryption with obfuscation, giving a false sense of security. Or none of the reports cover the session downgrade command, and the flag responsible for it. This work instead provides the first comprehensive and sound description and security evaluation of these protocols.

Security Analysis of Xiaomi Ecosystems. Xiaomi manages multiple ecosystems, including e-scooters, smartphones, smart home devices, and fitness trackers. In [11], the authors root a Xiaomi vacuum cleaning robot, inspect its internals, assess data privacy, and flash the robot with custom firmware. Another previous work [44] also finds several security issues with Xiaomi vacuum cleaners.

Several researchers [10, 15, 19, 49] highlight the limitations of the application-layer protocols run by Xiaomi Mi Band fitness trackers over BLE. These devices were found vulnerable to eavesdropping, man-in-the-middle, and impersonation. Using a fuzzing approach, X. Du et al. [13] find 95 vulnerabilities in the R1D Xiaomi router. Other Xiaomi IoT devices evaluated in the academic literature are Xiaomi smart speakers [21] and Xiaomi security cameras [43, 45].

BLE Misuse in Android. Researchers identified multiple flaws in Android BLE APIs. For example, Android saves Bluetooth keys in data structures shared among different apps [25, 41], allowing malicious apps to communicate illegitimately with paired devices. In [42], V. Toubiana et al. present a vulnerability, available from Android 6 to Android 11, that allows an Android app to perform a BLE scan without requiring location permission. Android applications may also misuse the BLE link-layer, allowing attackers to bypass encryption and authentication procedures [51]. In this paper, we focus on application-layer protocols instead and only utilize Android BLE APIs in our remote threat model.

Attacks on BLE Pairing. Several attacks over the years have targeted BLE link-layer pairing. In 2013, Crackle [32] broke the Just Works and Passkey modes of BLE Legacy pairing by brute-forcing their temporary key. In 2019, the KNOB [2] attack minimized the entropy of the encryption key in BLE Legacy pairing and Secure Connections, allowing for brute-force attacks on that key. In 2021, Method Confusion [48] performed a man-in-the-middle attack on BLE Secure Connections by separately pairing two devices in two different pairing modes. Xiaomi e-scooters do not utilize BLE link-layer pairing. Instead, we reverse-engineer and attack the proprietary Xiaomi Pairing phase (and Session) at the application-layer.

10 CONCLUSION

We present the first security evaluation of the proprietary security protocols employed by Xiaomi to protect its e-scooter ecosystem since 2016. We uncover and reverse-engineer four protocols using ad-hoc Pairing and Session mechanisms at the application-layer on top of an insecure BLE link-layer. We describe their (lack of) security properties.

We show four novel attacks to exploit protocols at the specification level requiring realistic and low-cost attacker models (i.e., a proximity-based adversary with a laptop or remote attacker who installed a malicious app on the victim's smartphone). The attacks enable stealing a software-locked and password-protected e-scooter from its owner or preventing the owner from using the e-scooter via Mi Home. The threats pivot on MP and SD attack strategies and are enabled by six severe root causes that we also uncover.

We open-source `E-Spoofer`, a toolkit implementing our attacks and offering RE utilities for the Xiaomi e-scooter ecosystem (e.g., protocol dissectors, Ghidra scripts, and Frida hooks). We successfully evaluate our attacks in eight relevant scenarios covering five e-scooter BLE subsystems and eight BLE firmware. We empirically demonstrate that our attacks have a critical impact on the Xiaomi ecosystem (e.g., all reversed protocols are affected by at least two of our four attacks), amounting to millions of exploitable devices.

We propose two practical, low-cost, and backward-compliant countermeasures to stop our attacks and release them in our toolkit. We propose Authorized Pairing Mode and Password-Protected Authenticated Pairing to fix the MP attacks and a script to stop the SD attacks by automatically patch the vulnerable e-scooter BLE firmware.

ACKNOWLEDGMENTS

Work funded by the the Air Force Office of Scientific Research under award number FA8655-20-1-7048 and European Union under grant agreement no. 101070008. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] The App Analyst. 2019. App Analysis: Bird. https://theappanalyst.com/bird.html/.
[2] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B. Rasmussen. 2019. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1047–1061. https://www.usenix.org/conference/usenixsecurity19/presentation/antonioli
[3] Ltd Beijing Xiaomi Mobile Software Co. 2022. Mi Home (Android). https://play.google.com/store/apps/details?id=com.xiaomi.smarthome.
[4] Ltd Beijing Xiaomi Mobile Software Co. 2022. Mi Home (iOS). https://apps.apple.com/us/app/mi-home-xiaomi-smart-home/id957323480.
[5] Brian Benchoff. 2019. Security Engineering: Inside the Scooter Startups. https://hackaday.com/2019/02/12/security-engineering-inside-the-scooter-startups/.
[6] BotoX. 2022. Xiaomi M365 Firmware Patcher. https://github.com/BotoX/xiaomi-m365-firmware-patcher.
[7] Buildxyz. 2020. nRFSec. https://github.com/buildxyz-git/nrfsec.
[8] CamiAlfa. 2017. M365 55AA BLE Protocol. https://github.com/CamiAlfa/M365-BLE-PROTOCOL.
[9] CamiAlfa. 2022. M365Downg. https://play.google.com/store/apps/details?id=com.m365downgrade.
[10] Marco Casagrande, Eleonora Losiouk, Mauro Conti, Mathias Payer, and Daniele Antonioli. 2022. BreakMi: Reversing, Exploiting And Fixing Xiaomi Fitness Tracking Ecosystem. *IACR Transactions On Cryptographic Hardware And Embedded Systems* 2022, 3 (2022), 330–366. https://doi.org/10.46586/tches.v2022.i3.330-366
[11] Daniel AW Dennis Giese. 2017. Unleash Your Smart-Home Devices: Vacuum Cleaning Robot Hacking. https://media.ccc.de/v/34c3-9147-unleash_your_smart-home_devices_vacuum_cleaning_robot_hacking.
[12] Piotr Dobrowolski. 2019. M365 5AA5 BLE Protocol. https://github.com/Informatic/py9b.
[13] Xuechao Du, Andong Chen, Boyuan He, Hao Chen, Fan Zhang, and Yan Chen. 2022. AflIot: Fuzzing On Linux-Based IoT Device With Binary-Level Instrumentation. *Computers & Security* 122 (2022), 102889. https://doi.org/10.1016/j.cose.2022.102889
[14] STM Electronics. 2022. ST-LINK Debugger V2. https://www.st.com/en/development-tools/st-link-v2.html.
[15] Hossein Fereidooni, Tommaso Frassetto, Markus Miettinen, Ahmad-Reza Sadeghi, and Mauro Conti. 2017. Fitness Trackers: Fit for Health but Unfit for Security and Privacy. In *2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*. 19–24. https://doi.org/10.1109/CHASE.2017.54
[16] Ian D. Foster. 2019. Xiaomi M365 Scooter Authentication Bypass. https://lanrat.com/xiaomi-m365/.
[17] Enrico Punsalang from InsideEVs. 2022. Segway-Ninebot Has Sold More Than One Million E-Scooters In China. https://insideevs.com/news/613420/segway-ninebot-one-million-sold-china/.
[18] VooDooShamane from Rollerplausch.com. 2022. MiDu Flasher. https://rollerplausch.com/threads/midu-flasher-st-link-downgrade-unbrick.5399/.
[19] Andrew Hilts, Christopher Parsons, and Jeffrey Knockel. 2016. Every Step You Fake: A Comparative Analysis of Fitness Tracker Privacy and Security. *Open Effect Report* 76, 24 (2016), 31–33.
[20] iResearch. 2018. iResearch Coverage On Xiaomi. http://www.iresearchchina.com/Upload/201808/20180824143739_3256.pdf.
[21] Li Lin, Xuanyu Liu, Xiao Fu, Bin Luo, Xiaojiang Du, and Mohsen Guizani. 2021. A Non-Intrusive Method For Smart Speaker Forensics. In *ICC 2021 - IEEE International Conference on Communications*. 1–6. https://doi.org/10.1109/ICC42927.2021.9500679
[22] MiEcosystem. 2019. Mijia BLE Libraries. https://github.com/MiEcosystem/mijia_ble.
[23] Sandeep Mistry. 2022. Noble NodeJS BLE Central Module (Abandonware). https://www.npmjs.com/package/@abandonware/noble.
[24] Daljeet Nandha and Florian Bruhin. 2022. EC MiAuth Library. https://github.com/dnandha/miauth.
[25] Muhammad Naveed, Xiaoyong Zhou, Soteris Demetriou, Xiaofeng Wang, and Carl Gunter. 2014. Inside Job: Understanding and Mitigating the Threat of External Device Mis-Bonding on Android. https://doi.org/10.14722/ndss.2014.23097
[26] BBC News. 2019. Scooters Hacked To Play Rude Messages To Riders. https://www.bbc.com/news/technology-48065432.
[27] NSA. 2022. Ghidra Reverse Engineering Suite. https://ghidra-sre.org/.
[28] OPENOCD. 2022. OPENOCD. https://openocd.org/.
[29] Ole André Vadla Ravnås. 2023. Frida. https://frida.re/.
[30] Reuters. 2015. Xiaomi-backed Chinese firm acquires iconic scooter maker Segway. https://www.reuters.com/article/us-ninebot-xiaomi-investment-idUSKBN0N60GN20150415.
[31] Pavel Revak. 2019. PySWD. https://github.com/cortexm/pyswd.
[32] Mike Ryan. 2013. Bluetooth: With Low Energy Comes Low Security. In *Proceedings of the 7th USENIX Conference on Offensive Technologies* (Washigton, D.C.). USENIX Association, USA, 4.
[33] ScooterHacking. 2022. ScooterHacking Firmware Toolkit. https://mi.cfw.sh/.
[34] ScooterHacking. 2022. ScooterHacking Github. https://github.com/orgs/scooterhacking/repositories.
[35] ScooterHacking. 2022. ScooterHacking Ninebot EC Protocol. https://wiki.scooterhacking.org/doku.php?id=nbdocs.
[36] ScooterHacking. 2022. ScooterHacking Utility App. https://play.google.com/store/apps/details?id=sh.cfw.utility.
[37] ScooterHacking. 2022. ScooterHacking Website. https://scooterhacking.org/.
[38] Nordic Semiconductor. 2021. nRF51822 Product Specification. https://infocenter.nordicsemi.com/pdf/nRF51822_PS_v3.4.pdf.
[39] Nordic SemiConductor. 2022. nRF51822 System-on-Chip. https://www.nordicsemi.com/Products/nRF51822.
[40] Dariusz Seweryn. 2022. RxAndroidBle Library. https://github.com/dariuszseweryn/RxAndroidBle.
[41] Pallavi Sivakumaran and Jorge Blasco. 2019. A Study of the Feasibility of Co-located App Attacks against BLE and a Large-Scale Analysis of the Current Application-Layer Security Landscape. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1–18. https://www.usenix.org/conference/usenixsecurity19/presentation/sivakumaran
[42] Vincent Toubiana and Mathieu Cunche. 2021. No Need to Ask the Android: Bluetooth-Low-Energy Scanning without the Location Permission. Association for Computing Machinery, New York, NY, USA, 147–152. https://doi.org/10.1145/3448300.3467824
[43] Zouheir Trabelsi. 2022. Investigating The Robustness Of IoT Security Cameras Against Cyber Attacks. In *2022 5th Conference on Cloud and Internet of Things (CIoT)*. 17–23. https://doi.org/10.1109/CIoT53061.2022.9766814
[44] Fabian Ullrich, Jiska Classen, Johannes Eger, and Matthias Hollick. 2019. Vacuums In Ihe Cloud: Analyzing Security In A Hardened IoT Ecosystem. In *13th USENIX Workshop on Offensive Technologies (WOOT 19)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/woot19/presentation/ullrich
[45] Mathy Vanhoef. 2021. Fragment and Forge: Breaking Wi-Fi Through Frame Aggregation and Fragmentation. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 161–178. https://www.usenix.org/conference/usenixsecurity21/presentation/vanhoef
[46] Nisha Vinayaga-Sureshkanth, Raveen Wijewickrama, Anindya Maiti, and Murtuza Jadliwala. 2022. An Investigative Study On The Privacy Implications Of Mobile E-Scooter Rental Apps. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (San Antonio, TX, USA). Association for Computing Machinery, New York, NY, USA, 125–139. https://doi.org/10.1145/3507657.3528551
[47] VirusTotal. 2022. Yara. https://virustotal.github.io/yara/.
[48] Maximilian von Tschirschnitz, Ludwig Peuckert, Fabian Franzen, and Jens Grossklags. 2021. Method Confusion Attack on Bluetooth Pairing. In *2021 IEEE Symposium on Security and Privacy (SP)*. 1332–1347. https://doi.org/10.1109/SP40001.2021.00013
[49] Jiliang Wang, Feng Hu, Ye Zhou, Yunhao Liu, Hanyi Zhang, and Zhe Liu. 2020. BlueDoor: Breaking the Secure Information Flow via BLE Vulnerability. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services* (Toronto, Ontario, Canada). Association for Computing Machinery, New York, NY, USA, 286–298. https://doi.org/10.1145/3386901.3389025
[50] Xiaomi. 2022. Xiaomi Bug Bounty Program On HackerOne. https://hackerone.com/xiaomi?type=team.

[51] Qingchuan Zhao, Chaoshun Zuo, Jorge Blasco, and Zhiqiang Lin. 2022. PeriScope: Comprehensive Vulnerability Analysis of Mobile App-Defined Bluetooth Peripherals. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security* (Nagasaki, Japan). Association for Computing Machinery, New York, NY, USA, 521–533. https://doi.org/10.1145/3488932.3517410

[52] Rani Idan (Zimperium). 2019. Don't Give Me A Brake – Xiaomi Scooter Hack Enables Dangerous Accelerations And Stops For Unsuspecting Riders. https://blog.zimperium.com/dont-give-me-a-brake-xiaomi-scooter-hack-enables-dangerous-accelerations-and-stops-for-unsuspecting-riders/.

APPENDIX

RE Methodology

We present the RE methodology that we employed to reconstruct the protocols described In Section 4. This methodology can be *reused* by other researchers to tackle similar RE efforts (e.g., reversing a proprietary and unknown application-layer protocol implemented on top of an insecure BLE link-layer). Specifically, we explain how we analyzed the Xiaomi and BLE traffic, the Mi Home app, and the scooter's BLE firmware.

Xiaomi and BLE Traffic. The BLE e-scooter exposes a GATT server with unknown characteristics used to exchange the proprietary P1, P2, P3, P4 payloads. We enumerate these characteristics with a GATT client program (e.g., `gatttool`). The e-scooter utilizes a Xiaomi custom GATT service (`0xFE95`), and its UPNP (`0x0010`) and AVDTP (`0x0019`) characteristics for Pairing and Session, and the Nordic UART service for the encrypted communication during Session. Then, we run multiple Pairing and Session phases using different combinations of e-scooter models, BLE subsystems, and firmware. We capture the generated BLE traffic (HCI-layer, including GATT) in dedicated pcap files. The pcap files are produced by our Android smartphone running Mi Home with Developer Options, and HCI Snooplog turned on. We open the pcaps in Wireshark with custom display filters to focus only on the proprietary application-layer payloads. We use Pyshark and Scapy to reverse the Xiaomi payloads and develop custom dissection classes capable of decoding and re-encoding the packets. For example, we developed the `PairChal` and `SessRand` classes to dissect P4 Pairing and Session security mechanisms. Table tab:appendix-opcodes lists BLe packets from P3 and P4 Pairing and Session phases.

Mi Home for Android. For Mi Home, we focused on its Android version because it is much easier to inspect and reverse than its iOS counterpart. We locate the Mi Home APK with `adbshellpmpathc om.xiaomi.smarthome`. We pulled It with `adb pull` and extracted its content, including the decompiled Smali and Java code, with `apktool` and `jadx`. We perform static analysis of the decompiled Java code, looking for API calls to cryptographic primitives and Android's BLE framework. In parallel, we use Frida and Objection for dynamic binary instrumentation of Mi Home. Frida requires a rooted phone and a Frida server application running on the phone. With our dynamic tests, we can list all the Classes involved with Mi Home, isolate the ones related to P1, P2, P3, and P3, hook them to observe their runtime execution, and reimplement some of their behaviors. For example, we found that Mi Home, during Pairing, calls `_m_j.fyp.O0000000` to perform ECDH, and, during Session, calls `_m_j.fys.O0000000` to perform the HMAC-based authentication and `_m_j.fyl.O0000000` to encrypt communications.

E-Scooter BLE Firmware. Reversing the e-scooter BLE firmware is challenging, as is a stripped binary with no debugging symbols. We obtain multiple firmware samples from different sources, i.e., ScooterHacking repositories, the Mi Home app storage, the Xiaomi backend, and by reading the BLE SoC memory at runtime via the ST-Link debugger. We statically analyze the firmware using Ghidra configured for ARM Cortex M0 little-endian. We also configure the Ghidra memory layout using the nRF51822 Product Specification 3.4 [38] from Nordic Semiconductors. In particular, we consult the instruction table to retrieve the memory addresses for instantiating the peripherals such as the FICR_UICR, POWER, CLOCK, and GPIO. We also use the ST-Link debugger to inspect the firmware at runtime using gdb, dumping its memory and flashing it with different BLE firmware versions.

Table 3: Mapping between the vulnerabilities (rows) and the presented attacks (columns). We put a ✓ if an attack exploits a vulnerabilit. Otherwise, we put an ✗. We split our two attacks, Malicious Pairing (MP) and Session Downgrade (SD), depending on their threat model, either proximity-based or remote.

	Proxim.		Remote	
Vulnerability	**MP**	**SD**	**MP**	**SD**
V1: Unauthenticated Pairing	✓	✗	✓	✗
V2: Unintentional Pairing mode	✓	✗	✓	✗
V3: Improper e-scooter passw. enfor.	✓	✓	✓	✓
V4: Unprotected sensitive memory	✓	✓	✓	✓
V5: Downgradable and insec. Session	✗	✓	✗	✓
V6: No BLE sec. despite device support	✗	✓	✗	✓

Table 4: BLE packets for the Xiaomi protocol P3 and P4. The first and second columns indicate the name we assigned to the packet and the protocol which uses them. The third and fourth columns specify who sends the packet and its content. The value "00X0" stands for an increasing counter (i.e., 0010, 0020, 0030, 0040) placed in a fragmented packet.

Packet	P	From	Content
SessReq	P3	App	`5AA5 0E 3D21 5D00 Serial`
SessReqOk	P3	Esc	`5AA5 0E 213D 5D00 Serial`
Comms	P3	App, Esc	`5AA5 Len From To Cmd Pay`
PairReq	P4	App	`A2000000`
PairReqOk	P4	Esc	`000000000200`
PairChal1	P4	Esc	`0010 01000000 Chal2Part`
PairChal2	P4	Esc	`0020 Chal2Part`
PairECStart	P4	App, Esc	`000000030400`
PairPubKey	P4	App, Esc	`00X0 PubKey4Part`
PairECEnd	P4	App	`000000000200`
PairSol	P4	App	`00X0 PairSol2Part`
PairSolAck	P4	Esc	`00000100`
PairOk	P4	App	`13000000`
PairOkAck	P4	Esc	`11000000`
SessReq1	P4	App	`24000000`
SessReq2	P4	App	`0000000B0100`
SessRand	P4	App, Esc	`0100 AuthChal`
SessAskRand	P4	Esc	`0000000C0200`
SessSol	P4	App, Esc	`00X0 AuthSol2Part`
SessOk	P4	Esc	`21000000`
Comms	P4	App, Esc	`55AB Len Count Encr Cksm`