USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on the Xiaomi E-Scooter Ecosystem

Marco Casagrande, *EURECOM;* Riccardo Cestaro, Mauro Conti, and Eleonora Losiouk, *University of Padua;* Daniele Antonioli, *EURECOM*

https://www.usenix.org/conference/vehiclesec26/presentation/casagrande

This paper is included in the Proceedings of the 4th USENIX Symposium on Vehicle Security and Privacy.

August 10–11, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-56-4

Open access to the Proceedings of the 4th USENIX Symposium on Vehicle Security and Privacy is sponsored by USENIX.

E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on the Xiaomi E-Scooter Ecosystem

Marco Casagrande
EURECOM, Biot, France

Riccardo Cestaro
University of Padua, Padua, Italy

Mauro Conti
University of Padua, Padua, Italy

Eleonora Losiouk
University of Padua, Padua, Italy

Daniele Antonioli
EURECOM, Biot, France

Abstract

The popularity of electric scooters has grown in recent years, both for personal use and through rental apps. E-scooters are typically managed via a mobile app and powered by internal components, such as the battery, the battery management system (BMS), the driving system (DRV), and the Bluetooth Low Energy system (BTS). Despite their safety implications, there is little research on the security and privacy of these proprietary and undocumented internals and their associated attacker models. To fill this gap, we present the first security and privacy assessment of the internal components of two popular Xiaomi e-scooters. We cover the M365 (2016) and Mi3 (2023) e-scooters, and their Mi Home companion app.

Via reverse-engineering (RE), we uncover four critical design vulnerabilities on the e-scooter internals, such as arbitrary code execution on the BMS. Based on these issues, we develop E-Trojans, five novel attacks that flash a malicious BMS firmware to exploit the e-scooter's internal components. The attacks can be performed physically via the debug port of the BMS, wirelessly over Bluetooth Low Energy (BLE), and remotely from a malicious app on the victim's phone. They have a critical and real-world impact as they violate the safety, security, availability, and privacy of e-scooters and their users. For instance, they can permanently damage the e-scooter battery by undervolting it below the safety threshold or track the user by utilizing e-scooter internal values as a fingerprint.

We implement our attacks and RE findings in a modular and low-cost toolkit. Our toolkit binary patches BMS firmware by adding malicious capabilities, such as disabling battery safety thresholds, which enables our attacks and the creation of new ones. We test our attacks on real M365 and Mi3 e-scooters, empirically confirming their effectiveness and practicality. We propose four practical countermeasures that improve the security, privacy, and safety of the Xiaomi e-scooter ecosystem and its millions of users. We responsibly disclosed our findings to Xiaomi, which acknowledged and addressed them.

1 Introduction

Electric scooters are pervasive embedded systems used for smart and sustainable mobility. Their market is valued at USD 37 billion, with an estimated annual growth of 9.9% [51, 52]. They carry sensitive data, such as geolocation and telemetry, and can be wirelessly controlled by the user from a mobile companion app. A vulnerable e-scooter poses serious security, privacy, and safety risks, such as theft by defeating the motor-locking mechanism or physical injuries by tampering with the braking system.

E-scooters have complex and proprietary *internals* typically including a battery and its management system, an electric motor and its controller, and a wireless module supporting a low-power communication such as Bluetooth Low Energy (BLE). Moreover, the architecture includes internal buses, such as Universal Asynchronous Receiver-Transmitter (UART) and Inter-Integrated Circuit (I2C), allowing the modules to exchange data.

E-scooters' internals represent an attractive *attack surface*. Prior research explored internal attack surfaces in domains such as automotive, targeting electric control units (ECUs) [12] and internal communication buses [30, 72], drones, flashing malicious flight controller firmware to remotely control them [55, 56], and laptops' batteries [41]. In the context of e-scooters, researchers focused on external attack surfaces, such as the proprietary over-the-air communication protocols [11] and e-scooter rental mobile apps [69].

However, the security and privacy of *e-scooters' internals* have received no attention from the scientific community and, despite the safety risks, there is no knowledge regarding the consequences of an attacker breaking into internals, like the battery management firmware. We fill this gap by presenting a security and privacy assessment of the internals of two popular e-scooters sold by Xiaomi and its subsidiary Segway-Ninebot, which are market leaders [17]. To systematically analyze the whole attack surface, we consider three real-world attacker models: physical, proximity, and remote. We target the *M365* and *Mi3* models, covering two e-scooter generations

from 2016 to 2024. These e-scooters are used by millions of people [62] who either own an e-scooter or rent it using services like Bird [6,71]. We also cover *Mi Home*, the Xiaomi e-scooter companion app available for Android and iOS.

We reverse-engineer (RE) the e-scooters' internals using static and dynamic techniques, including firmware disassembly and decompilation and packet analysis of internal and external communications. During our RE experiments we uncover *four design vulnerabilities* (V1–V4). For instance, the presence of an unsigned BMS firmware allowing for arbitrary code execution in case of a rogue firmware update. Being design flaws, they are effective on *any* M365 and Mi3 e-scooter. Other e-scooters, such as the Pro, Pro2, 1S, and Essential, are also vulnerable to our attacks because they share the same internals as the M365 and Mi3. Through RE, we also reveal new information about the e-scooter internals, such as how to disable the hardware safety protections in the BMS, prevent the battery from charging, and alter brake settings.

We consider three realistic attacker models: *physical*, *proximity*, and *remote*. The physical attacker has one-time access to the e-scooter and the BMS debug port, the proximity attacker communicates over BLE with a nearby e-scooter, and the remote attacker leverages a malicious BLE-enabled app installed on the victim's smartphone to remotely connect to the e-scooter.

We propose **E-Trojans**, *five novel attacks on the e-scooter internals*. Our attacks flash a malicious battery management system (BMS) firmware and demonstrate what an attacker can accomplish given arbitrary code execution on the BMS. For instance, we present an *overvoltage battery destruction* attack that permanently damages the battery and an *undervoltage battery ransomware* attack that irreversibly degrades the e-scooter's battery until the victim pays a ransom, even if the e-scooter is turned off and not plugged into a charger. Both attacks can cause hazards, including fire and explosions due to physical phenomena like battery thermal runaway and polarity inversion [40]. Moreover, we show a novel way to track an e-scooter and its driver using a fingerprint computed from the e-scooter's internals (e.g., motor serial number) and advertising it over BLE. Our attacks have a critical and large-scale impact on the Xiaomi e-scooter ecosystem, breaking the safety, security, and privacy of millions of e-scooters (and their users).

We release the `E-Trojans` toolkit, implementing the E-Trojans attacks using low-cost and open-source software and hardware. Our toolkit contains a binary patcher to generate malicious BMS firmware. This patcher can inject in the BMS a set of eleven (low-level and malicious) firmware capabilities required to deploy our attacks. Such capabilities include turning off safety-critical voltage protections, disabling firmware updates, and altering the e-scooter's braking behaviour. The firmware patcher allows to design and deploy new attacks (i.e., beyond the E-Trojans attacks) by picking different combinations of capabilities, making it a generalizable tool to assess the security of e-scooters. While our toolkit targets STM8 chips, it can be extended to others, like STM32, with low engineering effort [65].

We confirm that the attacks are practical and effective by ethically testing them on 365 and Mi3 e-scooters in a controlled environment. Among others, we manage to overheat the Mi3 battery above 80°C (i.e., the thermal stability threshold [47]), degrade the M365 battery's autonomy by 50% in less than four hours via undervoltage, and track the e-scooters using BLE. To fix the presented vulnerabilities and attacks, we propose *four countermeasures* (C1–C4). These add confidentiality and integrity to the BMS (controller) firmware and protect the internal (UART) bus against spoofing and Denial-of-Service (DoS). Our fixes are *backward-compatible* and *low-cost*, as they reuse available cryptographic primitives and require minimal computational overhead. We summarize our **contributions** as follows:

- We identify the lack of security and privacy assessments on e-scooters' internals and present an analysis of two popular Xiaomi e-scooters (M365 and Mi3). Via extensive RE, we uncover four critical design vulnerabilities, including arbitrary code execution on a BMS by flashing a malicious and unsigned firmware.
- We present five novel attacks on e-scooter internals, that we call E-Trojans. They explore the consequences of running a compromised BMS on the e-scooter. For instance, they exploit battery overvoltage and undervoltage to damage the e-scooter battery, or track an e-scooter via a fingerprint computed from the e-scooter's internal components. The attacks are practical and deployable by a real-world physical, proximity-based, or remote attacker.
- We release `E-Trojans`, a toolkit implementing our attacks with open-source software and cheap hardware. The toolkit is usable to further RE the proprietary Xiaomi e-scooter ecosystem, target other non-Xiaomi e-scooters, and compose new attacks.
- We deploy our five attacks on actual M365 and Mi3 e-scooters, empirically confirming their practicality and stealthiness. We propose four effective, low-cost, and legacy-compliant countermeasures to fix the E-Trojans vulnerabilities and attacks.

Responsible Disclosure

We followed standard practices for responsible disclosure by contacting Xiaomi and the manufacturers of the BMS chips we target: STMicroelectronics (ST) and Texas Instruments (TI). We notified Xiaomi in November 2023 via their official HackerOne bug bounty program [74]. Xiaomi assigned a Medium (5.8) CVSS severity score and closed our report as informative in March 2024. In April 2024, we contacted TI and ST. TI could not comment on the Xiaomi third-party

firmware and highlighted that their STM8L151K6 chip does not claim resistance to physical attack (we note that our attacks can also be conducted remotely and in BLE proximity. ST did not respond to our report.

After a second round of disclosure in June 2025, Xiaomi acknowledged our attacks, awarding us with the highest bounty for our tier and a Medium severity CVE. They also stated that the M365 and Mi3 models have reached the end of their lifecycle, as described in [75], and that the E-Trojans vulnerabilities have been mitigated in all subsequent Xiaomi electric scooter models, which now incorporate enhanced security measures. Hence, our findings concretely contributed to a more secure Xiaomi e-scooter ecosystem.

2 Background, Motivation, and Threat Model

In this section, we provide background information, motivate our work, and present our system and attacker models.

2.1 Background

Figure 1 shows the block diagram for a Xiaomi e-scooter. An e-scooter is typically composed of a Bluetooth Low Energy system (BTS), a Driving system (DRV), and a Battery Management system (BMS). The BTS provides low-power and reliable wireless communication, the DRV controls the electric motor, and the BMS manages the battery. Each system has a dedicated System-On-Chip (SoC) running closed-source and undocumented firmware. These three systems communicate using a proprietary protocol over a UART bus that provides a serial, asynchronous, and full-duplex digital communication channel with TX/RX wires.

Mi Home App. The user interacts with a Xiaomi e-scooter via the *Mi Home* app for Android or iOS. The e-scooter and Mi Home discover each other using BLE scanning and advertising, and communicate using a proprietary protocol on top of BLE [11]. The user logs in with their Xiaomi account on Mi Home, pairs the app and the e-scooter, and then uses the app to interact with the e-scooter. Once paired, the devices automatically connect when they are in BLE range without requiring user interaction. Mi Home offers valuable features for controlling the e-scooter, such as setting a password to lock it, performing over-the-air (OTA) firmware updates, and monitoring the battery status.

E-Scooter Internals. There is limited information about Xiaomi e-scooters' internals. In [11], the authors focus on external communications with Mi Home and do not discuss internal components. Research on the Xiaomi 1S e-scooter reports that Xiaomi encrypts the BTS, DRV, and BMS firmware using a custom version of the TEA cipher featuring a leaked symmetric encryption key [7], potentially allowing the creation of custom firmware. However, the BTS and DRV have been found to be protected by ECDSA-based signature verification and include a certificate chain with a public key used to validate integrity [43].

Firmware Update. During a firmware update, Mi Home transmits firmware to the BTS, which decrypts, verifies, and distributes them to the DRV and the BMS over the UART bus. The developers of the ScooterHacking Utility [59, 61] Android app reversed an old Xiaomi firmware update protocol to enable flashing a modded BTS, DRV, or BMS firmware on e-scooters, but did not publicly disclose the protocol details. Moreover, there is no information about recent BTS, DRV, and BMS firmware and their update protocols.

Battery. The M365 and Mi3 e-scooters are equipped with a lithium-ion (Li-Ion INR) battery pack. The battery contains ten individual cells in a 5x2 configuration: two parallel groups, each made of five cells connected in series. Their nominal range goes from 42V (100% charge) to 36V (0% charge).

2.2 E-Trojans Motivation

Our research is motivated by the severe *security*, *privacy*, and *safety* risks posed by vulnerabilities and attacks on e-scooters' internals. For example, a compromised BMS can hijack the lithium-ion battery or the driving subsystem to cause physical damage to an e-scooter and its owner. Existing research in this area is limited. The author of [41] compromised a laptop's battery management system without evaluating the feasibility of battery overcharging and overheating on a real device. In [15], the authors modified office printers' firmware but did not explore attacks on the battery.

While prior work on internal attack surfaces exists for automotive [29] and drones [8, 56], the *internal attack surfaces for e-scooters* is largely unexplored, with only a few papers studying their external threats. In [11], the authors reverse-engineered and compromised proprietary communication protocols used by Xiaomi e-scooters and Mi Home over BLE. In [9], researchers performed a threat analysis on the Xiaomi M365 and reproduced known attacks on its authentication. The authors of [28] investigated physical and DoS attacks on e-scooter sharing platforms. Privacy issues have also been found in e-scooters' apps [69] that shared user identity, location, and phone data with the manufacturer and third parties.

To address this gap, the paper explores the internal attack surface of an e-scooter and its exploitability using real-world physical, proximity-based, and remote attacker models.

2.3 Threat Model

Our system model follows the block diagram in Figure 1 and includes an e-scooter, a charger, a smartphone, and a user. The e-scooter is either owned by the user or rented from a fleet and runs up-to-date firmware. The smartphone is owned by the user and runs Mi Home. The user paired their Mi Home app and account with the e-scooter. Moreover, they activate

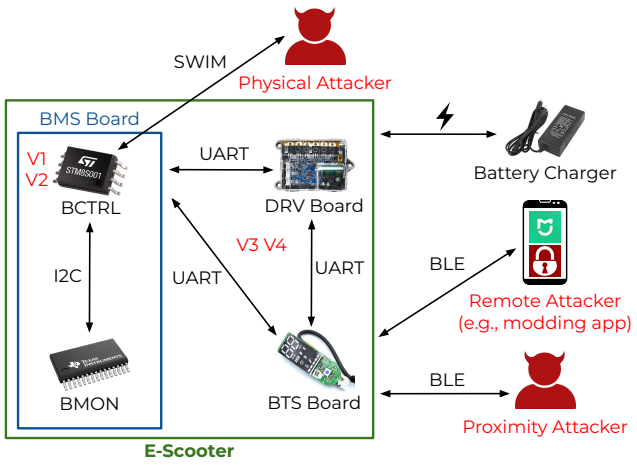

Figure 1: E-scooter block diagram and E-Trojans attacker models. The green rectangle shows e-scooter internals (BMS, DRV, and BTS boards, and UART and I2C buses). The blue rectangle shows BMS components (BCTRL and BMON connected via I2C). We consider: a physical attacker with access to the BMS debug (SWIM) port, a proximity-based attacker in the BLE range of the e-scooter and a remote attacker who installed a rogue app (e.g., an e-scooter modding app) on the victim's smartphone. We show four design vulnerabilities affecting the BCTRL (V1, V2) and the UART bus (V3, V4).

the software lock feature, allowing users to lock and unlock the e-scooter from the app.

As shown in Figure 1, we consider three attacker models: (i) a *physical attacker* with direct access to the e-scooter and the Single Wire Interface Module (SWIM) debug port of its BMS, (ii) a *proximity-based attacker* who interacts with the e-scooter wirelessly over BLE (e.g., using their device in BLE range with the victim's e-scooter); and (iii) a *remote attacker* who connects to the victim's e-scooter over the Internet through a malicious app installed on the victim's smartphone. These models systematically address the whole attack surface and map to real-world attack scenarios. For example, the proximity-based attacker can place a BLE device in a train station or a parking spot to target e-scooters in range, while the remote attacker can install a malicious BLE-enabled app on a victim's smartphone.

The attacker goal is to violate the *safety*, *security*, *privacy*, and *availability* of the e-scooter and its driver by attacking its internal components. For example, the attacker might want to attack the BMS or the DRV subsystems via a malicious firmware update over BLE. Achieving these goals leads to a large-scale and critical impact on the Xiaomi e-scooter ecosystem by affecting millions of e-scooters and their users. The attacker reuses tools, knowledge, and techniques from prior

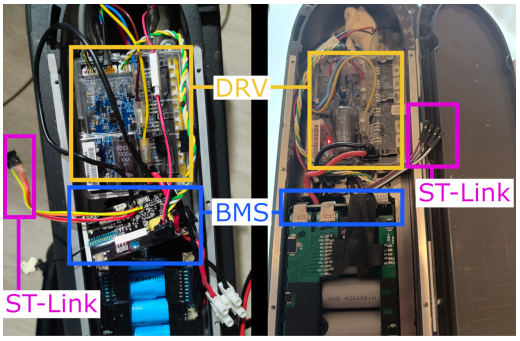

Figure 2: Disassembled M365 (left) and Mi3 (right). We color-coded the boxes to visualize the DRV (orange), the BMS (blue), and the soldered ST-Link wires (fuchsia).

work [11, 43, 59, 76]. For example, they can use the malicious BLE device and app from the `E-Spoofer` toolkit [10].

3 Xiaomi E-Scooters RE and Vulnerabilities

This section summarizes our RE findings on the M365 and Mi3 with an emphasis on their BMS. Then, it describes the four E-Trojans vulnerabilities. Refer to the Appendix for a description of our RE methodology, and more information on the internal (UART and I2C) communication protocols.

3.1 RE Findings

In 2023, we dedicated approximately nine months of RE effort on the Mi3 and M365 e-scooters. We picked these two e-scooters because the Mi3 was the latest e-scooter from Xiaomi, and the M365 was the most popular one. These models represent two different e-scooter generations. However, they share the same internal architecture, BCTRL firmware, and communication protocols over UART and I2C.

We disassembled the M365 and Mi3 e-scooters by tearing down their lower deck, gaining access to the DRV, BMS, UART bus, and battery pack, as shown in Figure 2. We extracted and inspected the BMS board of the e-scooters, finding a *battery controller (BCTRL)* and a *battery monitor (BMON)*. The BCTRL and BMON work together to ensure safe battery management. They communicate through an I2C bus, which is a synchronous two-wire serial interface (SDA and SCL) used for communication between integrated circuits.

Our security assessment focuses on three unsafe conditions that impact battery health and user safety: *overvoltage (OV)*, which can lead to overheating, fire, and thermal runaway; *undervoltage (UV)*, which can lead to polarity inversion and permanent battery damage; and voltage *imbalance* between battery cells, which contributes to both undervoltage and overvoltage, accelerating battery degradation [23, 31, 63]. We now detail new RE findings on the BCTRL and BMON of the

Mi3 and M365, as they are needed to describe the E-Trojans attacks and relative root causes.

BMON. The BMON is a TI BQ76930 chip [25]. It measures battery parameters (i.e., voltage and temperature) via dedicated sensors, reports faults to the BCTRL, and stores safety thresholds (i.e., OV and UV thresholds). These thresholds are kept in dedicated BMON registers and are configurable by the BCTRL with read and write commands sent using proprietary I2C messages. When a safety threshold is surpassed, the BMON relies on hardware protections, such as a hardware load balancer for the battery cells, or charge and discharge MOSFETs to stop charging and discharging. These protections are used to prevent or mitigate hazardous situations, like battery overvoltage or undervoltage. Being a battery monitoring integrated chip, the BMON does not run firmware or support hardware debugging [26].

BCTRL. The BCTRL is an ST STM8L151K6 chip [64]. It manages the battery's health with the BMON and communicates with the DRV and BTS. For example, it powers off the DRV and BTS when the battery undervolts or overvolts, initializes BMON registers during startup, and processes battery data received from the BMON. We identified the power supply (VDD), Single Wire Interface Module (SWIM), Ground (GND), and Reset (RST) pins used for powering, programming, and resetting the BCTRL. We soldered them to an *ST-Link v2* programmer for firmware extraction and debugging, finding *no hardware debug (ST-Link) protection*.

We RE the BCTRL firmware initialization and main loop by decompiling its stripped binary and debugging it via ST-Link. During initialization, the BCTRL loads its default settings, including serial number and firmware version. It also configures the BMON over I2C, including its overvoltage and undervoltage thresholds and the voltage balancing delta. In the main loop, the BCTRL firmware: communicates over UART with the BTS and DRV, reads the voltage of individual battery cells from BMON read-only registers, and checks these voltages against the overvoltage and undervoltage thresholds and the voltage balancing delta. If a safety check fails, the BCTRL sends an OV or UV fault message over UART, which powers off the BTS and DRV, enters a low energy consumption mode, and performs voltage load balancing on imbalanced cells.

Battery Overvoltage. A fully charged cell in a Xiaomi e-scooter battery pack has 4.2V nominal voltage. This is also the default value of the *overvoltage threshold*, stored in the BMON's `OV_TRIP` register and used to protect the battery. Overvoltage occurs when the voltage of a cell exceeds the overvoltage threshold and goes above 100% charging state. This is because 0% and 100% are relative values depending on the chemistry of the battery and not absolute ones. An overvolted cell is a serious safety hazard, capable of causing permanent battery damage, cell voltage imbalance, battery overheating, and even fire or an explosion [31]. In case of overvoltage, the BMON raises a fault (OV fault bit =1, `sys_stat` register) that can be cleared (bit set back to 0) by the BCTRL.

Battery Undervoltage. An e-scooter battery cell is considered discharged (0% charge) when its voltage is 3.6V, the nominal value defined by the manufacturer. However, cell voltage can drop as low as 2.75V, which is the default *undervoltage threshold*, below which the cell is considered undervolted. This threshold is stored in the BMON's `UV_TRIP` register and applies to all battery cells. Undervoltage reduces battery longevity and health [23,27,63], prevents safe recharging, and can result in safety hazards. For example, it can trigger polarity inversion, which can cause safety risks such as internal short circuits, voltage imbalance, swelling, fires, or an explosion. In case of undervoltage, the BMON raises a fault (UV fault bit =1, `sys_stat` register) and the fault can be cleared (bit set back to 0) by the BCTRL.

3.2 E-Trojans Vulnerabilities

Through RE, we uncovered *four design vulnerabilities* on the M365 and Mi3 internals:

V1: BCTRL firmware is weakly (or not) encrypted. An attacker can retrieve the BCTRL binary from Mi Home (at rest) or during a BLE firmware update (in transit). The M365 BCTRL firmware is not encrypted, while the Mi3 BCTRL can be decrypted through a leaked key [43].

V2: BCTRL firmware is unsigned. An attacker can modify a BCTRL firmware or craft a new one, and flash it on the BCTRL without having to authenticate it.

V3: UART bus lacks integrity protection, encryption, and authentication. An attacker on the UART bus can eavesdrop on the UART messages, replay or forge them, and impersonate or MitM the DRV, BMS, and BTS.

V4: UART bus lacks protection against DoS. An attacker accessing the UART bus can DoS the BMS, BTS, and DRV.

These issues stem from the *insecure design* of the Xiaomi e-scooter internals and firmware management, and are independent from the hardware and software implementation details of the e-scooters. V1–V4 exist in the M365, Mi3, and *all* other models that share the same BCTRL firmware and internal architecture, like the Pro, Pro2, 1S, and Essential. As a consequence, our attacks affect a wide range of e-scooters.

4 E-Trojans Attacks

We present *E-Trojans*, five novel attacks on the internals of Xiaomi e-scooters exploiting V1–V4 from Section 3.2:

1. Overvoltage Battery Destruction (OBD)
2. Undervoltage Battery Ransomware (UBR)

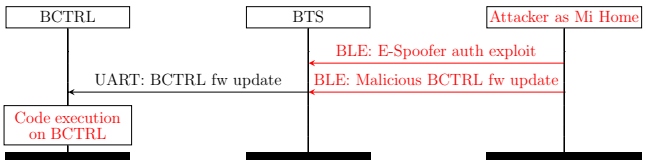

Figure 3: E-Trojans attack technique. The attacker, impersonating Mi Home, utilizes the Malicious Pairing or Session Downgrade technique from E-Spoofer [11] to authenticate to the BTS of the victim's e-scooter. Then, they perform a rogue firmware update that installs a malicious BCTRL, resulting in arbitrary code execution.

3. User Tracking via Internals (UTI)
4. Denial of E-Scooter Services (DES)
5. Password Leak and Recovery (PLR)

The attacker executes each E-Trojans attack by deploying a malicious BCTRL firmware on the victim's e-scooter. This firmware is specifically crafted to provide the attacker with arbitrary code execution and direct access to the e-scooter's internal attack surface. The delivery of the rogue BCTRL firmware can be achieved in multiple ways. For example, Figure 3 shows how to break authentication in the Xiaomi BLE proprietary protocol to flash BCTRL firmware over BLE. A physical attack with one-time access to the e-scooter can access the BCTRL debug port and flash firmware over SWIM. A software supply chain attack on the Mi Home firmware distribution server constitutes another viable remote attack vector. Table 4 in the Appendix maps the E-Trojans attacks and vulnerabilities.

Our attacks raise severe safety concerns, including the risk of fires and explosions, by overvolting and undervolting the e-scooter battery beyond the maximum overvoltage threshold or the minimum undervoltage threshold. We further discuss these safety hazards in Section 6.4. The attacks compromise the security of the Xiaomi e-scooter ecosystem by, for example, disabling firmware updates and spoofing internal components. They also violate privacy by tracking the e-scooter and its user, and break availability via DoS. Next, we describe each attack in detail.

4.1 Overvoltage Battery Destruction (OBD)

OBD exploits overvoltage to damage and destroy the e-scooter battery, resulting in a DoS attack which threatens the user's safety.

OBD employs a malicious BCTRL firmware to disable overvoltage and voltage balancing protections in the BMON, overvolting and ultimately destroying the e-scooter battery cells, as shown in Figure 4. OBD activates upon reaching the default overvoltage threshold of 4.2V, when the BMON

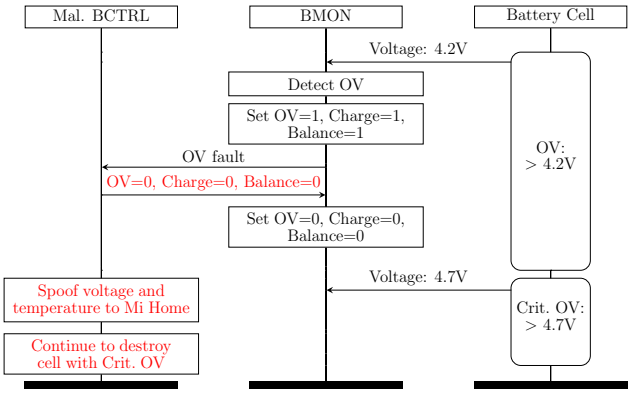

Figure 4: Overvoltage Battery Destruction (OBD). The Malicious BCTRL (the attacker) sets the OV threshold to the highest value (i.e., 4.7V). When the BMON detects overvoltage and tries to mitigate it, the attacker stops it by modifying the overvoltage fault bit, the MOSFET charge bit, and the voltage balancing bits. It also maintains stealthiness by spoofing fake sensor readings to Mi Home.

overvoltage safety protections are triggered (e.g., OV fault and stop charging). However, OBD defeats them by clearing the OV fault (OV fault bit =0, `sys_stat` register), keeping the charger connected to the battery (MOSFET charge bit =0, `sys_stat` register), and disabling load balancing (voltage balancing bits =0, `CELLBAL` registers).

This technique enables what we define as *critical overvoltage* (i.e., voltage above 4.7V). OBD remains undetected by spoofing voltage and temperature measurements to Mi Home. Since our firmware disables voltage balancing, the battery cells remain in an overvolted and unbalanced state that degrades and destroys them over time.

OBD reveals, for the first time, a *software-induced safety attack* on e-scooters, performed by a malicious BCTRL that can cause critical overvoltage without physical access or custom hardware.

4.2 Undervoltage Battery Ransomware (UBR)

UBR is an *undervoltage ransomware* that progressively and irreversibly damages the e-scooter's battery. As opposed to classical ransomware, which targets files, it asks for a ransom to stop ongoing physical damage to the e-scooter battery. It can also be deployed as a DoS attack that undervolts and destroys the battery without asking for a ransom.

UBR disables the BMON undervoltage and voltage balancing protections to undervolt the e-scooter battery cells, as shown in Figure 5. First, UBR prevents charging (MOSFET charge bit =1, `sys_stat` register), locks the e-scooter

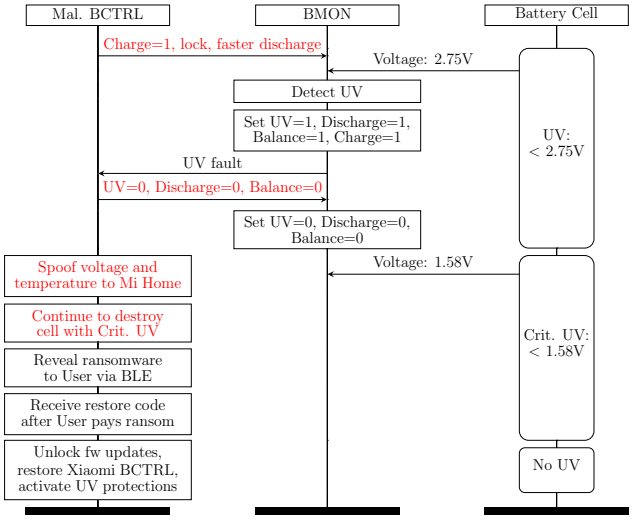

Figure 5: Undervoltage Battery Ransomware (UBR). The Malicious BCTRL (the attacker) sets the UV threshold to the lowest value (i.e., 1.58V), locks the e-scooter, and prevents it from charging (MOSFET charge bit). When the BMON detects undervoltage, the attacker breaks safety protections by altering the undervoltage fault bit, the MOSFET discharge bit, and the voltage balancing bits. UBR maintains stealthiness by spoofing sensor data, but, when triggered, will reveal its presence via BLE advertising. If the user pays the ransom, the ransom payment app submits the restore code that enables firmware updates and flashes the legitimate BCTRL firmware.

to keep it powered on, and forces discharging by ignoring sleep instructions and, optionally, turning on the headlight and taillight. When cell voltage reaches 2.75V, the BMON undervoltage safety protections activate (e.g., UV fault and voltage balancing). However, UBR turns them off by clearing the UV fault (UV fault bit =0, `sys_stat` register), restoring current flow from the battery to other internals (MOSFET discharge bit =0, `sys_stat` register), and disabling voltage load balancing.

This technique allows voltage to drop below 1.58V, putting cells into a condition that we call *critical undervoltage*. The ransomware maintains stealthiness by spoofing safe battery parameters to Mi Home until a programmable trigger activates. UBR then reveals the ransomware to the user, rebooting the e-scooter to broadcast a short link via BLE advertising (e.g., *t.ly/AaBbCc*). This link enables the victim to download an app to pay the ransom and see real-time battery degradation. To stop the damage, the victim has to transfer cryptocurrency to the attacker's crypto wallet.

Once the attacker receives the payment, they share with the user through the ransom app a restore code unique to

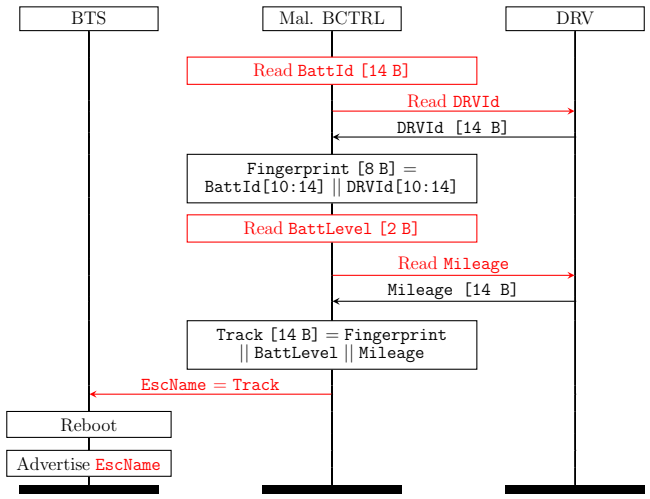

Figure 6: User Tracking via Internals (UTI). The Malicious BCTRL (the attacker) builds an 8-byte e-scooter fingerprint (`Fingerprint`) from the electric motor serial number. Then, they append other telemetry data like mileage and battery level to the fingerprint, creating a 14-byte tracking message (`Track`). Finally, they enforce a BLE name change to `Track`, allowing anyone to track the user and read their private data in real-time.

that e-scooter. The ransom app sends a special BLE packet containing that code to the e-scooter and then restores a legitimate BCTRL firmware. Undervoltage is fixed by reactivating the safety protections and charging the battery to a safe level.

UBR introduces the *first persistent, hardware-level ransomware* that monetizes physical degradation instead of data encryption. Attackers can rely on firmware vulnerabilities to cause irreversible hardware damage for profit.

4.3 User Tracking via Internals (UTI)

UTI is a privacy attack that tracks an e-scooter and its driver via a hardware-based fingerprint computed from the e-scooter's internals. Moreover, it can be used to leak the driver's data, such as travel mileage.

Figure 6 shows how UTI tracks users and leaks their e-scooter data. First, UTI reads the 14-byte battery serial number (`BattId`), and retrieves the 14-byte DRV serial number (Read `DRVId`) by impersonating the BTS and spoofing a read operation to the DRV. The attacker-controlled BCTRL builds a `Fingerprint` from the last 8 bytes of `DRVId` representing the year, month, and day of production, the revision, and the unit identifier (i.e., the nth unit produced that day).

UTI optionally retrieves via UART or I2C additional private data from the DRV, BMON, and BTS, like the real-time mileage and battery level (Read `BattLevel` and Read `Mileage`). Then, UTI appends the data to the fingerprint and

creates a 14-byte tracking message (`Track`). It sends a command to the BTS while spoofing the DRV to set the tracking message as the BLE name (`EscName = Track`). The e-scooter reboots and advertises its new name (`Track`) over BLE. The attacker can now track the user using pre-installed BLE sniffers at fixed locations, retrieving the user's position and telemetry data, which help infer travel route characteristics (e.g., if the user took a longer or shorter path).

UTI unveils a new *hardware-fingerprinting vector* based on internal serial numbers and telemetry leaked through manipulated BLE advertisements. Unlike traditional BLE-address-based tracking, this approach persists across factory resets and BLE randomization. The attack demonstrates that DRV internal identifiers can uniquely and permanently tag devices, introducing a previously unknown privacy risk.

4.4 Denial of E-Scooter Services (DES)

DES is a collection of eight DoS sub-attacks:

DES1: drops all UART and I2C packets sent to the BCTRL.

DES2: forces a supplier-only mode (SHIP mode) that disables the BMON by issuing a proprietary I2C command.

DES3: floods the UART bus with dummy packets.

DES4: periodically issues commands that reboot the e-scooter, causing a bootloop, or lock its motor, preventing the user from unlocking it.

DES5: disables battery charging unsetting the `canCharge` flag in the BCTRL flash.

DES6: exploits error handling and raises fake errors that cause the e-scooter to lock the motor or become unresponsive.

DES7: is a denial-of-sleep (DoSL) that keeps the BCTRL in a resource-intensive mode (other than low-energy mode) to discharge the battery faster.

DES8: alters braking settings and turns on/off regenerative braking (i.e., BCTRL slowing the e-scooter by converting kinetic energy into battery power, typically downhill).

DES expands the concept of DoS beyond network availability to embedded denial. By exploiting internal buses and control logic, it shows that low-cost firmware faults can disable propulsion, braking, charging, or entire subsystems. This attack highlights the need to examine *internal system communication (UART/I2C) as a DoS surface*.

4.5 Password Leak and Recovery (PLR)

PLR enables recovering the Mi Home password set by the victim user by leaking its hash from the e-scooter memory and brute forcing it offline.

On the Mi Home app, the user can set a password to prevent unauthorized access to their e-scooter via their smartphone. The password is stored on the phone, but we realized that its SHA256 hash is also stored in the DRV memory and is readable by the BTS via UART. PLR (BCTRL) spoofs the BTS to retrieve the password hash from the DRV, sets the e-scooter BLE name equal to the hash, and exfiltrates it using the e-scooter BLE name via BLE advertising.

PLR cycles through three BLE 14-byte BLE names containing portions of the hash. Each time it changes the BLE name, the e-scooter reboots. The password is six digits long (e.g., `123456`), and its search space is 10^6 [38]. Techniques such as password lists and rainbow tables can be used to brute force offline the hash to reconstruct the password. The attacker utilizes it to access the e-scooter and any other device using the same password (e.g., Android screen lock PIN).

PLR exposes a new leakage channel involving a secret stored in the DRV that can be exfiltrated over BLE advertisements. This highlights how the weak separation between the e-scooter subsystems allows recovery of authentication information, underlining the need for secure firmware boundaries and encrypted internal buses.

5 Implementation

We present `E-Trojans`, a new toolkit to conduct the E-Trojans attacks and create new ones. It includes code to send a rogue BCTRL firmware update and a binary patcher to build a firmware based on eleven malicious capabilities. While our attacks utilize fixed combinations (e.g., UTI uses C1, C2, C3, and C9), these eleven capabilities can be recombined to generate new attacks. The toolkit can be extended to work with other devices with internals and chips similar to the M365 and Mi3 e-scooters.

5.1 BCTRL Patching and Capabilities

The toolkit includes `payload-patcher.py`, a Python 3 script to binary patch a BCTRL firmware using STM8 assembly code. The script patches a vanilla BCTRL firmware, updating its CRC, and outputs a valid BCTRL firmware. Moreover, `payload-patcher.py` offers an interactive shell to craft the malicious BCTRL firmware. The user must select one of our five attacks and a supported e-scooter model (i.e., M365, Pro, Essential, Pro2, and Mi3). We support models *outside* of the M365 and Mi3 because they rely on the same BCTRL firmware and internal architecture. Thus, our malicious BCTRL is compatible with the Pro, Essential, and Pro2 models, and our techniques to flash the firmware (e.g., rogue over-the-air firmware update) remain effective.

The binary patcher includes the *eleven capabilities (C1–C11)* that compose the E-Trojans attacks, and can be combined in different ways to construct new attacks. Next, we describe each capability and how we use them.

C1 Disable Firmware Updates disables firmware updates on the BCTRL with a `canFwUpdate` flag (address 0x04CB) set to `false`. It changes the `HandleUART()` function (address 0x94C5) to accept or reject firmware update packets (starting with 0x07, 0x08, 0x09, or 0x0A), depending on `canFwUpdate`.

This flag is set to `true` when the BCTRL receives our new unlock packet (0xEE followed by the unlock code).

C2 Manage UART Bus sends arbitrary UART packets on the internal bus. It utilizes a USART1 peripheral mapped to RAM (address 0x5230). Due to V3, C2 crafts UART messages by altering fields such as the sender and receiver (e.g., 0x22 corresponds to BCTRL), packet type (e.g., read or write), command, payload, and CRC. It changes the `sendUART()` function (address 0xB06C) to send forged packets, stored from address 0xD3 onward. For example, Password Leak and Recovery uses C2 to spoof the BTS and retrieve the SHA256 hash of the e-scooter password from the DRV, but it can be used to spoof the DRV and BCTRL as well.

C3: Manage I2C Bus sends custom packets to the BMON through the I2C bus. These packets are used to read and write registers on the BMON and control it. For example, it can be used to read battery cells voltages from `VC1`, ..., `VC10` and alter the BMON undervoltage threshold in `UV_TRIP`. C3 rewrites `writeToI2C()` (address 0x90BE) to send forged packets on I2C.

C4: Modify Safety Thresholds alters the BMON undervoltage and overvoltage thresholds. It changes the BCTRL firmware boot logic, and initializes the BMON's `UV_TRIP` (address 0x923D) and `OV_TRIP` (address 0x923E) registers to 1.58V (lowest settable value) and 4.7V (highest settable value), or any other valid value. We remind that the default overvoltage and undervoltage safety thresholds are 2.75V and 4.2V. For example, C4 is used in OBD and UBR to force an overvoltage or undervoltage state that physically damages the battery, while not triggering any BMON safety protection.

C5: Disable Safety Protections disables extra safety protections in the BMON, allowing critical overvoltage (above 4.7V) and undervoltage (below 1.58V). The patch sets to 0 the BMON's OV fault, UV fault, MOSFET charge, and MOSFET discharge bits in the `sys_stat` register to prevent safety protections, such as preventing further charging, from triggering. It also rewrites the BCTRL error handler with NOPs to delete its OV and UV fault checks (address 0xB5A9) and ignore any fault message received from the BMON. For example, we use C5 in OBD to disable overvoltage protections in order to damage the battery via critical overvoltage.

C6: Disable Load Balancing speeds up cell deterioration, as load balancing is essential to compensate for voltage discrepancies across cells. It rewrites manageLoadBalancing() (address 0xA81A) to unset the three `CELLBAL` registers regulating balancing. For example, C6 is used by UBR to forcefully imbalance and kill one specific cell at a time.

C7: Disable Battery Charging disallows charging by rewriting `controlCharge()` (address 0x945C) and setting `canCharge` (address 0x42C) to `false`, regardless if the charger is connected. For instance, DES5 uses C7 to prevent the battery from charging.

C8: Fast Battery Discharge accelerates battery consumption by altering the BCTRL main loop. It sets to `false` the `sleepMode` variable (address 0x400) and forces the BCTRL to stay in a more resource-intensive mode. It can also turn on the e-scooter's headlight and taillight to further increase battery consumption. UBR uses C8 to accelerate battery depletion and critical undervoltage.

Table 1: Mapping between the E-Trojans attacks and the eleven capabilities offered by the `E-Trojans` binary patcher (`payload-patcher.py`). ✓*: OBD and UBR use either C4 or C5, depending on the attack scenario.

BCTRL Capability	OBD	UBR	UTI	DES	PLR
C1: Disable Fw Updates	✓	✓	✓	✓	✓
C2: Manage UART Bus	✓	✓	✓	✓	✓
C3: Manage I2C Bus	✓	✓	✓	✓	✗
C4: Modify Safety Thr.	✓*	✓*	✗	✗	✗
C5: Disable Safety Prot.	✓*	✓*	✗	✗	✗
C6: Disable Load Bal.	✓	✓	✗	✗	✗
C7: Disable Batt. Charge	✗	✓	✗	✓	✗
C8: Fast Batt. Discharge	✗	✓	✗	✓	✗
C9: Change BLE Adv.	✗	✓	✓	✗	✓
C10: Spoof Cell Voltage	✓	✓	✗	✗	✗
C11: Alter Brake Settings	✗	✗	✗	✓	✗

C9: Change BLE Advertising changes the e-scooter's BLE advertisements, including its BLE device name. It sends a proprietary message to the BTS containing the new BLE advertisement data. The message comprises a destination field (0x20), an opcode (0x50), and a payload containing the new advertisement. Once the message is received by the BTS, the e-scooter reboots and starts advertising over BLE with the new data. In UTI, the attacker tracks the e-scooter and its user via a custom BLE advertisement containing their fingerprint using C9.

C10: Spoof Cell Voltage fakes voltage measurements of battery cells to Mi Home to add stealthiness to an attack. It rewrites the BCTRL main loop to ignore the actual measurements from the BMON and reports arbitrary spoofed values. C10 sends these values to the BTS, which in turn sends them to Mi Home, which displays them to the user. Faking voltage allows UBR to hide the real voltage from the user while keeping undervolting the battery cells under the hood.

C11: Alter Brake Settings modifies brake-related settings managed by the BCTRL. It rewrites the default values for the brake lever pull (linear response between pull and braking force), brake phase current (regulates braking force), regenerative braking activation, and current raising coefficient (braking force in relation to speed). C11 can modify brake values in real-time depending on the context (e.g., downhill or crossing a road), maximizing the chances of injuring the user.

C1–C11 Usage. Table 1 shows how we combine the eleven capabilities to perform UBR, OBD, UTI, DES, and PLR. C1, C2, and C3 are common to all attacks, as each of them pre-

vents the user from reflashing the BCTRL, and send arbitrary packets over the UART and I2C bus (except PLR, which does not interact with I2C). C4, C5, C6, and C10 are leveraged by OBD and UBR to induce battery overvoltage and undervoltage by disabling voltage thresholds, protections, turning off load balancing, and sharing fake cell voltage. C7 and C8 are needed for UBR and DES, as they tamper with battery charge. C9 enables information disclosure from the e-scooter in UBR, UTI, and PLR, while C11 is required by DES8 to change downhill acceleration and mess with brake responsiveness.

5.2 UBR Ransom Android App and Backend

Our toolkit contains an Android app and a Django/MongoDB backend to test the UBR ransomware in a controlled, but realistic environment. These components generate unique ransom unlock codes, manage cryptocurrency payments, and handle BCTRL recovery.

ScooterToolkit. We developed *ScooterToolkit*, an Android app given to the victim so that they can pay the ransom in cryptocurrency and stop the undervoltage attack on their battery. The app is written in Kotlin and requires Bluetooth, Location, and Internet permissions. It initiates the BCTRL recovery when the back-end returns the unlock code that enables firmware updates (i.e., the user paid the ransom).

Django and MongoDB Backend. Using Django, we developed a RESTful web service that exposes two APIs to the ransom app through the APIView class. The `simulatePayment()` API simulates a user payment, setting the payment status to `Paid`. The `unlockFirmware(serial)` API requires the e-scooter's motor serial as its input, and, if the ransom has been paid, the unlock code is returned. We also created a model serializer that converts e-scooter objects into JSON to interface with our MongoDB database. This database stores the e-scooter's motor serial and model, 128-bit ransomware unlock codes, and payment status for each compromised device.

6 Evaluation

We evaluate the E-Trojans attacks (presented in Section 4) by using the `E-Trojans` toolkit (described in Section 5) on M365 and Mi3 e-scooters with up-to-date BTS, DRV, and BCTRL firmware. At the time of our experiments (2023), the M365 was the most popular model, while the Mi3 was Xiaomi's latest e-scooter. We also provide a video demonstration of UTI and DES4 at [19]. We describe our setup and results.

6.1 Evaluation Setup

We use the `payload-patcher.py` script in our toolkit to generate our malicious BCTRL firmware. The setup then varies depending on the attacker model of choice (physical, proximity-based, and remote). We test the physical attacker by connecting (only one time) an ST-Link debugger to the BCTRL SoC, as shown in Figure 2), and by directly flashing the malicious BCTRL firmware over the SWIM debug port.

For the proximity-based attacker, we run a Python script to connect over BLE from our machine (Dell Inspiron 15 3000 running a Linux-based OS) to a nearby e-scooter. We gain authorized access (e.g., via malicious pairing or session downgrade) and perform a rogue firmware update to flash the malicious BCTRL firmware.

As the remote attacker, we install the malicious *ScooterToolkit* Android app, found in our toolkit, on the victim's smartphone. We wait for the e-scooter to enter BLE range of the malicious app and then use the same technique as the proximity-based attacker to gain authorized access and perform a firmware update. We test the app on two different smartphones: a Realme GT (Android 13) and a OnePlus 3 (Android 9). We then set up each attack differently.

6.2 E-Trojans Attacks Setup

OBD Setup. To induce overvoltage, we connect two 30V power supplies in series and set them up to output 50V (i.e., above the 4.7V maximum overvoltage threshold). We monitor the voltage, temperature, and other battery parameters directly from the BMON by using our dynamic debugging setup. We plan to stop the experiment when a cell reaches either 50V or 80°C, to guarantee our safety (see Section 6.4).

UBR Setup. We configure UBR to activate when the e-scooter battery is charging at 5% or lower. We discharge the battery to 5% and connect the e-scooter to a charger, simulating a user who recharges their device overnight. Using our dynamic debugging setup, we observe and time the battery dropping from 5% charge to 0% (3.6V), and going beyond the minimum undervoltage threshold (below 1.58V). To assess how much permanent damage could be dealt, we maintain the battery in an undervolted state while monitoring parameters such as voltage, current, and temperature.

UTI Setup. We deploy BLE sniffers in four separate locations inside a private parking lot owned by our institution. These e-scooters travel on a predetermined route along the four locations, while the sniffers log their passing and the live data leaked by their BLE advertisements. The logs are then fed to a tracking system, updating the users' movements in real-time, and processed by scripts that attempt to de-anonymize them, based on where and for how long the users stopped during the current trip.

DES Setup. For each DES attack, we measure the time-to-failure and effect on user safety by observing their behaviour after the attack was triggered. We test DES8 on a mild downward slope while wearing protective equipment to mitigate fall injuries.

PLR Setup. We test the feasibility of retrieving the six-digit e-scooter password when deploying PLR. We consider the seven most common six-digit patterns [70] and randomly

Table 2: Evaluation results. All five attacks are effective on the M365 and Mi3 e-scooters. ✓[1]: OBD with dangerous overvoltage (4.2V). ✓[2]: UBR with dangerous undervoltage (1.58V).

Attack	M365	Mi3	Details
OBD	✓	✓[1]	Mi3 battery overheating in 5min
UBR	✓	✓[2]	M365 battery autonomy -50% in 3.5h
UTI	✓	✓	Tracks and de-anonymizes the user
DES	✓	✓	Sudden downhill deceleration
PLR	✓	✓	Leaks user password

generated seventy Xiaomi-compliant passwords (ten for each pattern). Then, we implement a rainbow table [35] and populate it with 3 million randomly generated Xiaomi-compliant passwords, simulating a basic attacker setup.

6.3 E-Trojans Attacks Results

Table 2 shows our evaluation results. The M365 and Mi3 e-scooters are vulnerable to the five attacks. Our experimental results demonstrate that overvoltage and undervoltage are relevant and practical threats, as the attacker can turn off safety protections on the BMS by programming the BMON from the rogue BCTRL firmware. However, they are less effective on the newer Mi3 e-scooter, whose DRV performs additional voltage checks and turns their power off if critical overvoltage or undervoltage is detected. We prove that the M365 and Mi3 e-scooters can be tracked over BLE, their password hashes can be leaked over BLE and brute forced offline, and their internals can be DoSed. We note that our empirical results represent a *lower bound* as other e-scooters might be vulnerable to our attacks For example, the Xiaomi Pro e-scooter runs BCTRL v1.2.6 (like the M365), and the 1S, Essential, and Pro2 run BCTRL v1.4.1 (like the Mi3). Next, we provide attack-specific insights.

OBD Eval. OBD overvolts and overheats the M365 battery within five minutes. We surpass the maximum overvoltage threshold and monitor the rising internal temperature of the battery. When it reaches the point of venting and electrolyte decomposition (i.e., 80°C), we halt the experiment. On the Mi3, we could only reach a dangerous overvoltage condition, as the DRV would power off the e-scooter when detecting critical overvoltage, preventing significant overheating.

UBR Eval. UBR irreversibly reduces the battery autonomy of the M365 by ∼50% in three hours and thirty minutes. We force a critical undervoltage state and stop the experiment after a few cells reach 0V (i.e., the most critical undervoltage condition). These cells now have limited charging capacity (i.e., 75%) and discharge 4x faster than healthy cells.

Similarly, we reduce the battery autonomy of the Mi3 by ∼10% in ten hours, spending six of them depleting the battery to 0% (three hours for the M365). We could not drop below the minimum undervoltage threshold due to the DRV independently measuring an abnormal voltage in the motor (RAM register with offset 0x47) and raising *Error 24 - Supply Voltage out of range*. As such, the Mi3 is better protected than the M365 against UBR, even though it still sustained significant damage.

UTI Eval. UTI successfully tracks users, exfiltrates their private data over BLE, and de-anonymizes them. We identify the two e-scooters from their fingerprint (e.g., motor serial number), allowing us to track users across the four areas where we deployed the BLE sniffers. Our tracking system, for example, reports up-to-date mileage (e.g., 0x01B1 represents 433km) and battery charge (e.g., 0x2D represents 45%), while identifying the locations where the user stopped along a predefined route.

We confirm that, by leaking the battery charge, total mileage, last travel time and mileage, and average speed, we can infer where they stopped, for how long, and whether they recharged the e-scooter (e.g., at home). We also verify that the user is still trackable after an e-scooter factory reset.

DES Eval. All DES attacks render the e-scooter unresponsive and not drivable within milliseconds. DES1 and DES3 cause the DRV to raise *Error 21 - No Communication with BMS*, halting the motor and powering off the e-scooter after ten seconds. DES2 cuts power from all internals, permanently powering off the e-scooter. DES4 prevents battery charging regardless of the charger hardware or e-scooter status (e.g., powered on/off). DES5 induces an endless lock state or reboot cycle, rendering the e-scooter unable to move or power off.

DES6 raises *Error 23 - Internal BMS not activated*, which induces a constant beeping noise, and *Error 24 - Supply Voltage out of range*, which powers off the e-scooter. DES7 ignores all sleep cycles even when the e-scooter is off, thus accelerating battery consumption. DES8 arbitrarily disables and re-enables regenerative braking, causing sudden and unpredictable deceleration and acceleration, putting a downhill driver at risk of falling off and injuring themselves. It also makes it harder to initiate braking by altering the brake lever responsiveness and modifies braking force by increasing or decreasing the brake current. By setting these parameters incorrectly on purpose, the attacker can generate too high a current and destroy the BMS board.

The DES attacks can target a variety of internal components and deny them. To restore the e-scooter to its original state, the user needs to flash the original BMON firmware via debug ports without the option of an over-the-air firmware update. This operation is complex, as it requires disassembling the entire battery compartment.

PLR Eval. PLR leaks the e-scooter password hash and enables offline brute forcing. We sniff three rotating BLE advertisements broadcasted by a compromised e-scooter, each containing part of the hash. Since the e-scooter resets after changing its advertisement, we rotate it in rapid succession, tricking the user into thinking a single reboot happened instead of three. We also confirm that retrieving the e-scooter

password from the hash is feasible. Our non-optimized rainbow table cracks the seventy passwords, following the most common six-digit patterns, in less than one second on average.

6.4 Experimental Safety Concerns

According to our experiments, OBD and UBR can be used to set a battery on fire and cause it to explode due to *induced hazards of overvoltage and undervoltage, and stealthiness*. We emphasize that the attacks are feasible from any distance, as shown in our evaluation of physical, proximity-based, and remote attackers, and that OBD and UBR do not require a charger. These aspects further increase the safety implications of the attacks. Next, we summarize the experimental evidence we collected to prove our claims.

During our UBR tests, we undervolt the battery *below the lowest BMS undervoltage threshold* (i.e., below 1.58V down to 0V). This dangerous condition is complex for users to detect, as the BCTRL spoofs valid battery levels to Mi Home and the user. During the OBD tests, we overvolt the battery *above the highest BMS overvoltage threshold* (i.e., above 4.7V up to 4.9V). The battery reached 80°C, whereas the recommended temperature range is between 0 °C and 45°C.

7 Countermeasures

We discuss how to address the E-Trojans attacks by design with *four fixes (F1–F4)*. Each countermeasure addresses one of the four vulnerabilities (i.e., F1 fixes V1 and so on) presented in Section 3.2.

F1: *Encrypt the BCTRL firmware with TEA*. The BCTRL firmware should be encrypted at rest (e.g., when stored by Mi Home) and in transit (e.g., during a BLE firmware update). We recommend reusing the custom Xiaomi implementation of the TEA block cipher already used to encrypt the DRV firmware (since v0.1.7).

F2: *Sign and verify the BCTRL firmware with ECDSA*. The BCTRL firmware should be digitally signed by Xiaomi and verified by the BCTRL to prevent rogue firmware updates. We recommend reusing ECDSA, which is FIPS-compliant [46] and already employed to sign/verify the firmware of the DRV (since DRV v0.1.7) and BTS (since BTS v1.5.2).

F3: *Protect the UART bus with SCP03*. The UART communication should be encrypted, authenticated, and integrity-protected. We suggest using standard embedded protocols such as *Secure Channel Protocol 03 (SCP03)* [49]. SCP03 provides confidentiality, integrity, authenticity, and replay protection using a lightweight scheme based on pre-shared symmetric keys, authenticated encryption, and AES. Since SCP03 is transport-agnostic, it can be adapted to UART with low engineering effort by adding message framing and flow control.

F4: *Protect the UART bus with rate limiting*. The UART bus should also be protected against DoS attacks. We recommend implementing an efficient rate-limiting mechanism for UART, such as the leaky bucket algorithm [34], where excess packets are discarded if the incoming packet rate exceeds the outgoing rate. Combining F4 with F3 enhances DoS protection, as F3 discards messages failing the integrity checks.

The fixes are *practical* and *backward-compatible*. They introduce negligible overhead, as they leverage lightweight cryptography (e.g., TEA and SCP03), while providing needed security guarantees, including confidentiality, authentication, and DoS protection. For instance, encrypting the 14KB BCTRL firmware with TEA would require ~135ms on Xiaomi's 16MHz BTS (i.e., nRF51822). They are backward-compatible with the firmware and internals of the e-scooters.

8 Related Work

E-Scooters' Security and Privacy. Research on e-scooters has studied their proprietary protocols and rental ecosystems, but no work has focused on their *internals*. A recent paper analyzed the Xiaomi e-scooter ecosystem, identifying issues in the proprietary BLE protocols used by the e-scooter and Mi Home [11]. Researchers identified vulnerabilities in the Bird e-scooter sharing platform [3] and discovered MitM attacks on the rental e-scooters of the Lime company [44]. Other studies focused on the privacy of user-related data in e-scooter rental Android apps [69] and the safety risks of riding e-scooters [37]. In 2019, researchers from Zimperium revealed flaws in the M365 locking system that could halt a running e-scooter [76]. The same year, Lanrat found that a M365 e-scooter did not enforce authentication over BLE [21].

E-Scooters Modding. There is a vibrant e-scooter modding community around the ScooterHacking [60] website. The community released useful tools such as ones to interact with Xiaomi e-scooters [58, 59] and perform firmware updates [7]. Other modding tools, such as [57], focus on e-scooter performance by, for example, increasing the maximum speed value. Our toolkit focuses on the previously unexplored BCTRL firmware and can add new malicious capabilities to the e-scooter BMS.

Attacks on Battery. The likelihood of battery overheating, thermal runaway, overcharging, and mechanical damage in electric vehicles has been investigated in [16, 32, 42]. Researchers have proposed a battery drain attack on gas and electric vehicles [13], an electromagnetic interference attack against automotive batteries [67], and a physical access rogue firmware update to a Tesla BMS [29]. Theoretical vulnerabilities in MacBook batteries, capable of overheating the

laptop, have been discussed in [41]. The E-Trojans attacks are *orthogonal* as they target e-scooters' battery and BMS.

Attacks on Embedded Firmware. Several papers focused on malicious embedded firmware, but none discussed a malicious BCTRL firmware causing battery overvoltage and undervoltage, or tracking e-scooter users. Researchers attacked embedded firmware on programmable logic control (PLC) [22], object trackers [53], printers [15], mice [39], OS management systems [24], mobile bootloaders [50], network equipment [14], botnets [4] and industrial wrenches [48].

Attacks on Battery-Powered Embedded Systems. Security research on battery-powered embedded systems has identified vulnerabilities across various domains. Automotive researchers exploited the insecure CAN bus by compromising an Electronic Control Unit (ECU) [30]. The authors of [2] document several CAN bus vulnerabilities, including the lack of authentication or traffic encryption, while [1] proposes an intrusion detection system in a vehicle's CAN bus to detect attacks such as battery degradation and explosion. Similar works have exposed wireless charging stations and EV infrastructure to attacks such as MitM, DoS, and privacy leakage [5, 33, 45, 54, 73]. Studies on drones have revealed issues in firmware updates, mobile apps, and wireless traffic [8, 18, 36, 56].

9 Conclusion

E-scooters are interesting attack targets as their exploitation can result in security, privacy, and safety issues. This paper focuses on their internals, which have received limited attention so far, mainly due to their complexity and proprietary nature. We focus on two popular Xiaomi e-scooter models, the M365 and Mi3. We RE their BMS, DRV, BTS subsystems, and their UART and I2C communication channels. We uncover four critical design flaws (V1–V4) enabling arbitrary code execution on the BCTRL firmware via a rogue firmware update over BLE. We use this attack primitive to design five novel attacks we call E-Trojans. Our attacks are executed with an attacker-controlled device in BLE proximity to the e-scooter or via a malicious app installed on the victim's phone.

OBD puts the battery in an overvoltage state by disabling the BMS overvoltage protections and damages it until destruction without being noticed by the end user. UBR disables undervoltage protections on the BMS and forces the battery into an undervoltage state, progressively and silently damaging it. Then, it asks the user for a ransom. UTI generates a unique e-scooter fingerprint from the e-scooter internals and broadcasts it over BLE to track the e-scooter and its user. It can optionally exfiltrate telemetry data. DES is a collection of attacks capable of DoS-ing the e-scooter's internal systems and communication in several ways.

We develop `E-Trojans`, a new toolkit that can conduct the E-Trojans attacks and create new ones using a binary patcher script supporting eleven malicious capabilities (C1–C11). We carried out the attacks on actual devices, proving their real-world impact and practicality. We discuss four fixes (F1–F4) to address the attacks at the design level in an efficient and backward-compatible way. We responsibly disclosed our contributions to Xiaomi, who acknowledged them, assigned us a CVE and the highest bounty for our category, and, most importantly, fixed the issues on newer e-scooter models.

Acknowledgments

This work is funded by the European Union – NextGenerationEU and by the 2023 STARS Grants@Unipd programme (Acronym and title of the project: "PatchThemAll: A Virtualization-Based Approach for Distributing Android Security Patches on Any Custom AndroidOS"). Besides funding this work, the above funding source had no involvement in the conduct of the research and in the preparation of the article. This work was partially supported by the French National Research Agency (ANR) under the France 2030 label, with the REV (ANR-22-PECY-0009) and NF-HiSec (ANR-22-PEFT-0009) research grants. We thank Professor Aurélien Francillon for his assistance in reverse-engineering the internals of the Xiaomi e-scooter.

Availability

We provide our toolkit and attack demos at https://doi.org/10.5281/zenodo.20529875. We release it as open-source, but, for safety reasons, we redacted the code responsible for generating malicious BMS firmware, which took months of RE to develop. The toolkit implements the E-Trojans attacks and shows our full attack chain in a video demonstration. It also contains a patched BCTRL firmware that blocks BMS firmware updates to prevent our attacks and our RE Ghidra projects for the M365 and Mi3 BMS firmware. Additionally, we release the undervoltage battery ransomware proof of concept, including a modified e-scooter firmware, an Android ransom payment app, and a Django/MongoDB backend managing the payment.

References

[1] S. Sai Akhil, K. Dhananjay Rao, T. Mansa, S. Tharun, and B. Dilip. A Brief Review of Cyber Attacks on Electric Vehicle Battery Management System. In *Innovations in Energy Management and Renewable Resources*, pages 87–98, 2024.

[2] Emad Aliwa, Omer Rana, Charith Perera, and Peter Burnap. Cyberattacks and Countermeasures for In-Vehicle Networks. *ACM Computing Surveys*, 54(1), 2021.

[3] App Analyst. App Analysis: Bird. https://theappanalyst.com/bird.html/, 2019.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the Mirai Botnet. In *26th USENIX security symposium (USENIX Security 17)*, pages 1093–1110, 2017.

[5] Richard Baker and Ivan Martinovic. Losing the Car Keys: Wireless PHY-Layer Insecurity in EV Charging. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 407–424, 2019.

[6] Brian Benchoff. Security Engineering: Inside the Scooter Startups. https://hackaday.com/2019/02/12/security-engineering-inside-the-scooter-startups/, 2019.

[7] BotoX. Xiaomi M365 Firmware Patcher. https://github.com/BotoX/xiaomi-m365-firmware-patcher, 2022.

[8] Pedro Cabrera. Parrot Drones Hijacking. https://www.rsaconference.com/Library/presentation/USA/2018/parrot-drones-hijacking, 2018.

[9] Louis Cameron Booth and Matay Mayrany. IoT Penetration Testing: Hacking an Electric Scooter, 2019.

[10] Marco Casagrande. E-Spoofer (GitHub). https://github.com/Skiti/ESpoofer, 2024.

[11] Marco Casagrande, Riccardo Cestaro, Eleonora Losiouk, Mauro Conti, and Daniele Antonioli. 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*, 2023.

[12] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *20th USENIX security symposium (USENIX Security 11)*, 2011.

[13] Kyong-Tak Cho, Yuseung Kim, and Kang G Shin. Who Killed my Parked Car? *arXiv preprint arXiv:1801.07741*, 2018.

[14] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A Large-scale Analysis of the Security of Embedded Firmwares. In *23rd USENIX security symposium (USENIX Security 14)*, pages 95–110, 2014.

[15] Ang Cui, Michael Costello, and Salvatore Stolfo. When Firmware Modifications Attack: A Case Study of Embedded Exploitation. In *NDSS*, 2013.

[16] Yan Cui, Jianghong Liu, Xin Han, Shaohua Sun, and Beihua Cong. Full-scale Experimental Study on Suppressing Lithium-ion Battery Pack Fires from Electric Vehicles. *Fire Safety Journal*, page 103562, 2022.

[17] Univ Datos. Folding Electric Scooter Market: Current Analysis and Forecast (2023-2030). https://univdatos.com/report/folding-electric-scooter-market/, 2022.

[18] Eddy Deligne. ARDrone Corruption. https://link.springer.com/article/10.1007/s11416-011-0158-4, 2011.

[19] E-Trojans. User Tracking and DoS demonstration on the Xiaomi Mi3 e-scooter. https://www.youtube.com/watch?v=Y2R46yeCXOQ, 2024.

[20] Alexey Esaulenko. Ghidra STM8 Processor Module (GitHub). https://github.com/esaulenka/ghidra_STM8, 2021.

[21] Ian D. Foster. Xiaomi M365 Scooter Authentication Bypass. https://lanrat.com/xiaomi-m365/, 2019.

[22] Luis Garcia, Ferdinand Brasser, Mehmet Hazar Cintuglu, Ahmad-Reza Sadeghi, Osama A Mohammed, and Saman A Zonouz. Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit. In *NDSS*, pages 1–15, 2017.

[23] Rui Guo, Minggao Ouyang, Languang Lu, and Xuning Feng. Mechanism of the Entire Overdischarge Process and Overdischarge-induced Internal Short Circuit in Lithium-ion Batteries. *Scientific Reports*, page 30248, 2016.

[24] Duha Ibdah, Nada Lachtar, Abdulrahman Abu Elkhail, Anys Bacha, and Hafiz Malik. Dark Firmware: a Systematic Approach to Exploring Application Security Risks in the Presence of Untrusted Firmware. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 413–426, 2020.

[25] Texas Instrument. 6 to 10-Series Cell Li-Ion and Li-Phosphate Battery Monitor. https://www.ti.com/product/BQ76930, 2022.

[26] Texas Instrument. BQ769x0 Datasheet. https://www.ti.com/lit/ds/symlink/bq76930.pdf, 2022.

[27] Texas Instruments. How To Protect 48-V Batteries from Overcurrent and Undervoltage. https://www.ti.com/lit/an/snoaa65/snoaa65.pdf, 2020.

[28] Gizay Kisa Isik, Theo Tryfonas, and George Oikonomou. E-scooter Sharing Platforms: Understanding Their Architecture and Cybersecurity Threats. In *2023 IEEE*

26th International Conference on Intelligent Transportation Systems (ITSC), pages 5909–5916, 2023.

[29] Patrick Kiley. Reverse Engineering the Tesla Battery Management System to Increase Power Available. https://www.youtube.com/watch?v=UV2zvgyIF0I, 2020.

[30] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental Security Analysis of a Modern Automobile. In *2010 IEEE symposium on security and privacy*, pages 447–462. IEEE, 2010.

[31] Berkeley Lab. Overcharge Protection Prevents Exploding Lithium Ion Batteries IB-3263. https://ipo.lbl.gov/lbnl3263, 2016.

[32] Binghe Liu, Yikai Jia, Chunhao Yuan, Lubing Wang, Xiang Gao, Sha Yin, and Jun Xu. Safety Issues and Mechanisms of Lithium-ion Battery Cell Upon Mechanical Abusive Loading: A Review. *Energy Storage Materials*, pages 85–112, 2020.

[33] Jianwei Liu, Xiang Zou, Leqi Zhao, Yusheng Tao, Sideng Hu, Jinsong Han, and Kui Ren. Privacy Leakage in Wireless Charging. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[34] Dimitris Logothetis and K Trivedi. The Leaky Bucket as a Policing Device: Transient Analysis and Dimensioning. In *INFOCOM*, volume 94, 1994.

[35] Charles Lu and Jialin Ding. Rainbow Table Attack on 6-Digit PINs (GitHub). https://github.com/clu8/RainbowTable, 2015.

[36] Aaron Luo. Multidimensional Attack Vectors and Countermeasures. https://infocondb.org/con/def-con/def-con-24/drones-hijacking-multi-dimensional-attack-vectors-and-countermeasures, 2016.

[37] Qingyu Ma, Hong Yang, Alan Mayhue, Yunlong Sun, Zhitong Huang, and Yifang Ma. E-Scooter safety: The Riding Risk Analysis Based on Mobile Sensing Data. *Accident Analysis and Prevention*, 2021.

[38] Philipp Markert, Daniel V Bailey, Maximilian Golla, Markus Dürmuth, and Adam J Aviv. This Pin Can Be Easily Guessed: Analyzing the Security of Smartphone Unlock Pins. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 286–303. IEEE, 2020.

[39] Jacob Maskiewicz, Benjamin Ellis, James Mouradian, and Hovav Shacham. Mouse Trap: Exploiting Firmware Updates in USB Peripherals. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, 2014.

[40] Carla Menale, Stefano Constà, Vincenzo Sglavo, Livia Della Seta, and Roberto Bubbico. Experimental Investigation of Overdischarge Effects on Commercial Li-Ion Cells. *Energies*, 15(22), 2022.

[41] Charlie Miller. Battery Firmware Hacking. *Black Hat USA*, pages 3–4, 2011.

[42] Shravan Murlidharan, Varsha Ravulakole, Jyothi Karnati, and Hafiz Malik. Battery Management System: Threat Modeling, Vulnerability Analysis, and Cybersecurity Strategy. *IEEE Access*, 13:37198–37220, 2025.

[43] Daljeet Nandha. Exploring Xiaomi's New Firmware Security Measures. https://robocoffee.de/?p=193, 2022.

[44] BBC News. Scooters Hacked To Play Rude Messages To Riders. https://www.bbc.com/news/technology-48065432, 2019.

[45] Tao Ni, Xiaokuan Zhang, Chaoshun Zuo, Jianfeng Li, Zhenyu Yan, Wubing Wang, Weitao Xu, Xiapu Luo, and Qingchuan Zhao. Uncovering User Interactions on Smartphones via Contactless Wireless Charging Side Channels. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 3399–3415. IEEE, 2023.

[46] NIST. FIPS 186-5 – Digital Signature Standard (DSS). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf, 2023.

[47] Gabriel Oltean, Nareerat Plylahan, Charlotte Ihrfors, Wei Wei, Chao Xu, Kristina Edström, Leif Nyholm, Patrik Johansson, and Torbjörn Gustafsson. Towards Li-Ion Batteries Operating at 80 °C: Ionic Liquid versus Conventional Liquid Electrolytes. *Batteries*, 4(1), 2018.

[48] PCmag. Network-Connected Torque Wrench Used in Factories Is Vulnerable to Ransomware. https://www.pcmag.com/news/network-connected-torque-wrench-used-in-factories-is-vulnerable-to-ransomware, 2024.

[49] Global Platform. SCP03. https://globalplatform.org/specs-library/secure-channel-protocol-03-amendment-d-v1-2, 2020.

[50] Nilo Redini, Aravind Machiry, Dipanjan Das, Yanick Fratantonio, Antonio Bianchi, Eric Gustafson, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. BootStomp: on the Security of Bootloaders in Mobile Devices. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 781–798, 2017.

[51] Grand View Research. Electric Scooters Market Size, Share & Trends Analysis Report (2024-2030). https://www.grandviewresearch.com/industry-analysis/electric-scooters-market, 2024.

[52] Grand View Research. Micro-mobility Market Size, Share & Trends Analysis Report By Vehicle Type (Electric Kick Scooters, Electric Skateboards, Electric Bicycles), By Battery, By Voltage, By Region, And Segment Forecasts, 2025-2030. https://www.grandviewresearch.com/industry-analysis/micro-mobility-market-report, 2025.

[53] Thomas Roth, Fabian Freyer, Matthias Hollick, and Jiska Classen. AirTag of the Clones: Shenanigans with Liberated Item Finders. In *2022 IEEE Security and Privacy Workshops (SPW)*, pages 301–311. IEEE, 2022.

[54] Khaled Sarieddine, Mohammad Ali Sayed, Sadegh Torabi, Ribal Attallah, Danial Jafarigiv, Chadi Assi, and Mourad Debbabi. Uncovering Covert Attacks on EV Charging Infrastructure: How OCPP Backend Vulnerabilities Could Compromise Your System. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, page 977–989, 2024.

[55] Nico Schiller, Merlin Chlosta, Moritz Schloegel, Nils Bars, Thorsten Eisenhofer, Tobias Scharnowski, Felix Domke, Lea Schönherr, and Thorsten Holz. Drone Security and the Mysterious Case of DJI's DroneID. In *NDSS*, 2023.

[56] Moritz Schloegel and Nico Schiller. Unchained Skies: A Deep Dive into Reverse Engineering and Exploitation of Drones. https://mschloegel.me/slides/slides_recon23_drone_security.pdf, 2023.

[57] ScooterHacking. ScooterHacking Firmware Toolkit. https://mi.cfw.sh/, 2022.

[58] ScooterHacking. ScooterHacking (GitHub). https://github.com/orgs/scooterhacking/repositories, 2022.

[59] ScooterHacking. ScooterHacking Utility App. https://play.google.com/store/apps/details?id=sh.cfw.utility, 2022.

[60] ScooterHacking. ScooterHacking Website. https://scooterhacking.org/, 2022.

[61] ScooterHacking. ScooterHacking Utility Homepage. https://utility.cfw.sh/, 2023.

[62] Ma Si. Segway-Ninebot Eyes Bigger Market Share in Country. https://global.chinadaily.com.cn/a/202204/28/WS6269ee14a310fd2b29e59d1f.html, 2022.

[63] Shashank Sripad, Sekar Kulandaivel, Vikram Pande, Vyas Sekar, and Venkatasubramanian Viswanathan. Vulnerabilities of Electric Vehicle Battery Packs to Cyberattacks on Auxiliary Components. *arXiv preprint arXiv:1711.04822*, 2017.

[64] STMicroelectronics. Ultra-low-power 8-bit MCU with 32 Kbytes Flash, 16 MHz CPU, integrated EEPROM. https://www.st.com/en/microcontrollers-microprocessors/stm8l151k6.html, 2018.

[65] STMicroelectronics. Migration of Applications from the STM8L and STM8S Series to the STM32C0 Series Microcontrollers. https://www.st.com/resource/en/application_note/an5775-migration-of-applications-from-the-stm8l-and-stm8s-series-to-the-stm32c0-series-microcontrollers-stmicroelectronics.pdf, 2022.

[66] STMicroelectronics. ST Visual Develop. https://www.st.com/en/development-tools/stvd-stm8.html, 2024.

[67] Marcell Szakály, Sebastian Köhler, Martin Strohmeier, and Ivan Martinovic. Assault and Battery: Evaluating the Security of Power Conversion Systems Against Electromagnetic Injection Attacks. *arXiv preprint arXiv:2305.06901*, 2023.

[68] ST-Link V2. ST-Link In-Circuit Debugger and Programmer for STM8 and STM32. https://www.st.com/en/development-tools/st-link-v2.html, 2024.

[69] Nisha Vinayaga-Sureshkanth, Raveen Wijewickrama, Anindya Maiti, and Murtuza Jadliwala. 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*, page 125–139, 2022.

[70] Ding Wang, Qianchen Gu, Xinyi Huang, and Ping Wang. Understanding Human-Chosen PINs: Characteristics, Distribution and Security. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS'17)*, 2017.

[71] Tech We Want. Who Makes Bird and Lime Scooters? https://techwewant.com/this-is-who-makes-bird-lime-and-jump-scooters-review-b3f6be32221e, 2018.

[72] Marko Wolf, André Weimerskirch, and Christof Paar. Security in Automotive Bus Systems. In *Workshop on Embedded Security in Cars*, pages 1–13. Bochum, 2004.

[73] Yi Wu, Zhuohang Li, Nicholas Van Nostrand, and Jian Liu. Time to Rethink the Design of Qi Standard? Security and Privacy Vulnerability Snalysis of Qi Wireless Charging. In *Annual Computer Security Applications Conference*, pages 916–929, 2021.

[74] Xiaomi. Xiaomi Bug Bounty Program (HackerOne). https://hackerone.com/xiaomi, 2024.

[75] Xiaomi. Xiaomi – Trust Center EOL Product List. https://trust.mi.com/misrc/updates/iot, 2025.

[76] Rani Idan (Zimperium). 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/, 2019.

Appendix

RE Challenges and Methodology

It took us around nine months to RE the M365 and ES3 internals and the Mi Home mobile app. This process was challenging for several factors. First, the e-scooter ecosystem is sophisticated, combining black-box hardware components with mobile application logic and opaque operations performed by the Xiaomi backend. Second, we started with no prior knowledge of the BCTRL firmware.

We had to physically inspect hardware components to identify them, and consult their datasheet before we could properly RE the BCTRL source code. The firmware lacks debug symbols and documentation, requiring extensive manual analysis of the decompiled code. Third, our experimental setup is non-trivial. It includes soldering wires to connect them to debug interfaces, installing proprietary software for runtime debugging, unbricking and re-flashing the e-scooters multiple times while testing the malicious BCTRL firmware. We also had to work within strict safety guidelines since we were messing with the voltage of dangerous lithium-ion batteries.

Our RE methodology combines static and dynamic analysis techniques, applied across different parts of the Xiaomi e-scooter ecosystem. We now describe each technique and then discuss how this methodology can be generalized to other e-scooter models and manufacturers.

Firmware static analysis. We used `binwalk` to scan the firmware and see if they were encrypted (i.e., entropy analysis) or contained certificates and public keys. Appended to the encrypted firmware, we found ECDSA signatures that we verified with `openssl`. We decrypted the BTS and DRV firmware using TEA and a leaked Xiaomi symmetric key (i.e., `UPDKEY=FE801CB2D1EF41A6A41731F5A06824F0`) [7].

We decompiled the BCTRL firmware using `Ghidra`. Since the BCTRL runs on an STM8L151K6 and `Ghidra` does not natively support the STM8 instruction set architecture, we installed the STM8 processor module extension [20]. We selected STM8 16-bit Big Endian as our Language and Compiler Specification and memory mapped the firmware.

We identified the RAM (0x0000 to 0x07FE), EEPROM (0x1000 to 0x13FF), FLASH (0x8000 to 0xFFFF), USART and I2C communication interfaces by consulting the microprocessor's datasheet. Then, we statically reversed the firmware, naming functions and variables based on their behavior. We started our RE from functions referencing the USART and I2C interfaces, tracing them back to the main loop. We also searched for values used by the internal protocols, like the UART packet header (i.e., 0x5AA5).

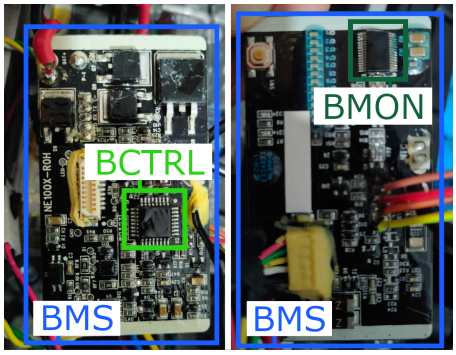

Figure 7: M365 BMS components on the M365. The BCTRL STM8 SoC (left) and the BMON BQ73930 SoC (right).

Firmware dynamic analysis. To perform the dynamic experiments, we disassembled the e-scooter chassis (as shown in Figure 2), protecting the BMS, DRV, and battery pack. Then, we extracted the BMS board and identified the BCTRL and BMON SoCs (as shown in Figure 7). The BCTRL exposed unlocked debugging ports, connectable through a *ST-Link v2* [68] debugger/programmer. After figuring out from the datasheet the pin layout, which was unknown for the ES3, we soldered four wires to the BCTRL's VDD, SWIM, GND, and RST pins. We used ST-Link onboard `gdb` server to debug a running BCTRL firmware through the *ST Visual Develop* IDE provided by ST [66].

We debugged the firmware at runtime with common practices. We set breakpoints and watchpoints to read and write to memory and inspect the CPU state. For instance, we set a breakpoint at 0xB06C to capture inbound UART packets and at 0x00D3 for outbound ones. Similarly, we set a breakpoint at 0xA24B to capture inbound I2C packets and we read the latest cell voltage measurement in the `X` register.

Local and backend traffic analysis. We captured local (BLE) traffic between Mi Home and an e-scooter by enabling *Android HCI snooplog* in the Developer Options. We opened the btsnoop port (8872) on our rooted smartphones, allowing us to monitor live BLE traffic through an interface in `Wireshark`. We retrieved the Xiaomi pairing key by performing session downgrade or malicious pairing as in [11].

Then, we triggered a BCTRL firmware update, using a script to decrypt the BLE traffic, from which we extracted the stripped BCTRL firmware. We eavesdropped on the backend (Wi-Fi or cellular) traffic during a legitimate Xiaomi firmware update using mitmproxy. To do so, we installed their CA certificate on our smartphone, set a proxy, and monitored the requests and responses from Mi Home. We also discovered

that the Xiaomi backend authenticates a firmware download request using its MD5 hash. This confirms that our BTS, DRV, and BCTRL binaries are legitimate and up-to-date.

Mi Home and ScooterHacking Android apps analysis. We studied the *closed-source* and *undocumented* Mi Home and ScooterHacking Utility Android apps to RE the Xiaomi proprietary firmware update protocol and the BLE packets that convey data from the BCTRL to the app. We performed static analysis using `JADX` and discovered that the code managing the communication with the e-scooter is obfuscated. We manually RE the decompiled code, finding the function responsible for generating BLE traffic, crypto primitives (encryption, decryption, authentication via challenge-response), and downloading BTS, DRV, and BCTRL firmware from the Xiaomi cloud. We utilized `Frida` for dynamic binary instrumentation to check functions' inputs and outputs and modify them at runtime during our RE experiments.

Methodology generalization. Our RE methodology is generalizable to other e-scooters since most use similar architectures. Our static and dynamic firmware analysis approach applies to any embedded firmware, which all feature RAM, EEPROM, FLASH memory, and internal UART and I2C bus. We target a systemic issue of battery management systems in embedded systems, showing how their components are not secure against an internal attacker. The same applies to the analysis we deploy on mobile e-scooter companion apps.

I2C and UART Binary Protocols

We RE the proprietary UART protocol used by the BCTRL, DRV, and BTS. The BCTRL uses a USART1 peripheral mapped to RAM (address 0x5230) to read and write UART. A UART packet has a constant 2 bytes header (0x5AA5), 2 bytes denoting the packet length, 2 bytes specifying the sender and receiver (e.g., 0x22 corresponds to BTS), 2 bytes identifying the packet type (e.g., read or write), 2 bytes indicating the command to execute, an N-byte payload, and a 2-byte CRC.

The BCTRL and BMON communicate using a proprietary protocol over I2C, which is neither encrypted nor authenticated. I2C is a synchronous serial digital protocol running on two wires (SDA, SCL) that supports master and slave devices, message addressing, and framing. I2C has a higher bandwidth than UART. The BCTRL can directly read and write the BMON registers via I2C. For example, we can set the overvoltage threshold by writing into `OV_TRIP` or read cell voltages via `VC1`, ..., `VC10`.

Table 3 presents a selection of UART and I2C commands we utilize in our attacks. We represent UART commands with their Xiaomi proprietary format and the I2C commands by their target register and accepted values. The *Lock*, *Reboot*, and *Shutdown* commands target the DRV and affect the e-scooter by preventing its movement, rebooting it or turning it off. We send the *Read PwdHash*, *WritePwdHash*, and *ReadDRVSerial* to the DRV to leak private or confidential data, like the hashed password or the DRV serial number.

Table 3: UART and I2C commands (column one) and content (column two). We represent UART commands as Xiaomi-compliant HEX packets, for example Change Adv includes two 1-byte fields (0x21 and 0x50) and the new advertising data. We represent I2C commands through the registers (R) they read or write. For example, the Set UVT command writes a value between 0 and 255 to R[0x9].

Src, Dst	Command	Content
UART bus		
BTS, BCTRL	Fw Update Init	0x220700‖ size
BTS, BCTRL	Fw Update Chunk	0x220800‖chunk
BTS, BCTRL	Fw Update Chksm	0x220900‖chksm
BTS, BCTRL	Fw Update End	0x220A00
DRV, BTS	Change Adv	0x2150‖advdata
BTS, DRV	Lock	0x20037001
BTS, DRV	Reboot	0x20037801
BTS, DRV	Power OFF	0x20037901
BTS, DRV	Read DRV Serial	0x20011006
I2C bus		
BCTRL, BMON	Read Voltage	0x0C-0x1F
BCTRL, BMON	Set OVT	0x09‖value
BCTRL, BMON	Set UVT	0x0A‖value
BCTRL, BMON	No Balancing	0x01‖0x00, 0x02‖0x00
BCTRL, BMON	Set Ship Mode	0x04‖0x01, 0x04‖0x02

We interact with I2C by calling `writeToI2C(register, value)` and `readFromI2C(register)`. We *Read Voltage* from the `VC1-10` registers (address 0x0C to 0x1F). We *Set OVT* and *Set UVT* by writing to the `OV_TRIP` and `UV_TRIP` registers (address 0x9 and 0xA). For example, writing 0 to `UV_TRIP` sets the overvotlage threshold to 1580mV, its lower possible value. We force *No Balancing* on the battery by writing 0 to the `CELLBAL1` and `CELLBAL2` registers.

Mapping Attacks and Vulnerabilities

Table 4 shows the mapping between the E-Trojans attacks and vulnerabilities.

Table 4: Mapping between the five attacks and the four vulnerabilities. All attacks exploit V1, V2, and V3. Only the DES attacks exploit V4.

E-Trojans Attack	V1	V2	V3	V4
Overvoltage Battery Destruction (OBD)	✓	✓	✓	✗
Undervoltage Battery Ransomware (UBR)	✓	✓	✓	✗
User Tracking via Internals (UTI)	✓	✓	✓	✗
Denial of E-Scooter Services (DES)	✓	✓	✓	✓
Password Leak and Recovery (PLR)	✓	✓	✓	✗

USENIX VehicleSec '26 Artifact Appendix: E-Trojans: Ransomware, Tracking, DoS, and Data Leaks on the Xiaomi E-Scooter Ecosystem

Marco Casagrande
EURECOM, Biot, France

Riccardo Cestaro
University of Padua, Padua, Italy

Mauro Conti
University of Padua, Padua, Italy

Eleonora Losiouk
University of Padua, Padua, Italy

Daniele Antonioli
EURECOM, Biot, France

A Artifact Appendix

A.1 Abstract

E-Trojans is a toolkit that implements five attacks on the Xiaomi M365 and ES3 (Mi3) electric scooters. We release a set of malicious BCTRL firmware that can be flashed on the e-scooter to alter its behaviour. For example, it can infect the e-scooter with a ransomware that induces undervoltage to permanently damage the battery. Each malicious firmware adds multiple malicious capabilities that we implement on the BMS after RE the stock Xiaomi firmware. The firmware speaks UART and I2C to communicate with the other entities on the UART (i.e., BTS and DRV) and I2C (i.e., BMON) bus. We also offer a proof-of-concept Android app that simulates ransom payment and its backend server, that includes a database and two web APIs. The toolkit works with minimal resources: a computer that supports BLE, Python (+ libraries), Java (+ libraries), MongoDB, Django, an Android phone (no root required), and a Xiaomi e-scooter (M365 or ES3).

A.2 Description & Requirements

The artifact consists of five folders.

Binary Tools. Contains Python scripts to analyzie firmware. No special requirements for testing. The binary-diff.py script identifies differences between two e-scooter binary firmware. This tool diffs firmware from different e-scooter models. It also helps reverse-engineering more recent and/or malicious firmware versions by directly comparing them to a baseline version. The binary-patcher script generates custom firmware compatible with most Xiaomi e-scooters (only M365 and ES3 have been properly tested). We also inlcude two legitimate BCTRL binaries for testing.

BMS Ghidra. Contains the reverse-engineered Ghidra projects for the Xiaomi 365 and ES3 BMS firmware. Requires Ghidra, and the installation of the STM8 Processor Modules (https://github.com/esaulenka/ghidra_STM8). The file can simply be opened and studied in Ghidra.

Malicious Payloads. Contains the malicious BMS firmware payloads. Requires an actual Xiaomi 365 or ES3 e-scooter to test. The payloads are firmware binary files that can be flashed on the e-scooter by our Ransom App (also released in this repository) or any other third-party app that allows for firmware updates, such as M365 DownG (https://play.google.com/store/apps/details?id=com.m365downgrade). As soon as the firmware is flashed on the e-scooter, the attack begins automatically. **FLASHING OUR FIRMWARE WILL LIKELY DAMAGE THE E-SCOOTER'S BATTERY AND SHOULD ONLY BE DONE ON TEST DEVICES. RESTORING IT WILL ALSO REQUIRE A COMPLICATED HARDWARE SETUP INVOLVING A ST-LINK DEBUGGER. DO IT AT YOUR OWN RISK.**

Ransom App. Contains the Android app that deploys our attacks, including the Undervoltage Battery Ransomware. Requires an Android smartphone. The app can scan for nearby Xiaomi e-scooters, flash a malicious firmware on them, perform our E-Trojans attacks, and restore an undervolted battery.

Ransom Server. Contains the files for the ransomware webserver and its database, to be used together with the ransom app. The webserver stores ransomware unlock keys and submits them to the ransom app if the user has the correct password (obtained by paying our ransom).

A.2.1 Security, privacy, and ethical concerns

Flashing our malicious firmware on an e-scooter leads to the safety hazards described in our paper, including battery overvoltage, undervoltage, rsik of fire and explosion, DoS, and permanent damage to the battery's health. If the reviewers want to test the attacks, they should contact us authors first. Experimenting with the is a safety-risk and should not be taken lightly. For this reason, we redacted parts of the malicious payload's code, making it not usable in this state.

This also means that people can not just clone our repo and perform dangerous attacks in the wild. They will need months of reverse-engineering. The other folders do not include dangerous code. They can be installed and ececuted freely. The ransom app and server perform basic non-malicious utility operations, but do not run any ransomware. The ransomware is contained inside the malicious firmware payload only.

A.2.2 How to access

We provide full and public access to our artifact in the following Zenodo DOI (https://doi.org/10.5281/zenodo.20529875).

A.2.3 Badges

We request the Available, Functional, and Reproduced badge.

A.2.4 Hardware dependencies

An Android smartphone. Bluetooth does not work on emulators, so emulated devices will have limited functionality. Optionally, if testing the attacks, a throwaway Xiaomi 365 or ES3 (Mi3) electric scooter.

A.2.5 Software dependencies

Standard requirements: Python and dependencies, Ghidra (with STM8 Processor Modules).

A.2.6 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Installation of software dependencies is straightforward and described in the repo. There are no particular challenges.

A.3.2 Basic Test

Clone the Github repo and check all files are there.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Our toolkit can diff Xiaomi BCTRL firmware, and patch it to create malicious versions.
(C2): Our toolkit contains the reverse-engineering Ghidra projects for the Xiaomi M365 and ES3 BCTRL firmware.
(C3): Our toolkit contains the malicious BCTRL firmware that can be flashed into a real M365 and ES3 e-scooter.
(C4): Our toolkit contains the infrastructure for the ransomware attack, including an Android app and webserver.

A.4.2 Experiments

(E1): BCTRL Firmware Diff [2 human-minutes].
Preparation: Open the *Binary Tools* folder.
Execution: Run `python binary-diff.py` to diff two BCTRL firmware versions. You can use default files, or modify them within the script.
Results: Binary diffs are shown in the console.
(E2): BCTRL Firmware Patching [5 human-minutes].
Preparation: Open the *Binary Tools* folder.
Execution: Run `python payload-patcher.py` to create a new patched and malicious firmware from the list. In the menu, we suggest to select e-scooter version=1 or version=6, and attack=4 or attack=5. We redacted the code for the other listed attacks for safety reasons.
Results: A new firmware is created in the folder.
(E3): Ghidra Reverse-engineering [15 human-minutes].
Preparation: Install Ghidra and the STM8 Processor Modules (https://github.com/esaulenka/ghidra_STM8). Open the *BMS Ghidra* folder.
Execution: Open the M365 or ES3 project files with Ghidra.
Results: The files will be correctly analyzed. Function names will be shown, our comments in relevant parts of the code (e.g., firmware updates) will be visible, and the memory map will be interpreted correctly (e.g., RAM, ROM, and Peripherals).
(E4): Malicious Payloads [10 human-minutes].
Preparation: Open the *Malicious Payload* folder.
Execution: Check the presence of the malicious BCTRL firmware. Check the User Tracking + Denial-of-Service attack video (https://www.youtube.com/watch?v=Y2R46yeCXOQ).
Results: The malicious firmware has been released and there is video proof of it working.
(E5): Ransomware App and Webserver [10 human-minutes].
Preparation: Open the *Ransom App and Ransom Server* folders and use Python.
Execution: Deploy them by following the instructions in the repo. First, install the app on the Android phone, scan for nearby Xiaomi e-scooters, and see if there are any results (of course, there might not be Xiaomi e-scooters nearby). Second, install the webserver and its requirements.
Results: Android app and Python scripts have been released and work.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.