Secure-by-Design Prototyping of an IoT Access-Control System

Oliver Vainikko[1], Ulrich Norbisrath[1], Ruben Jubeh[2]
[1]Department of Computer Science
University of Tartu
Tartu, Estonia
{oliver.vainikko|ulrich.norbisrath}@ut.ee
[2]Faculty of Computer Science and Mathematics
OTH Regensburg
Regensburg, Germany
ruben.jubeh@oth-regensburg.de

***Abstract*—Rapid Internet of Things prototyping frameworks accelerate hands-on teaching and proof-of-concept development, but often normalize insecure deployments, such as plaintext messaging, shared credentials, and weak over-the-air update protection. This paper examines IoTempower, an educational framework for ESP8266 and ESP32 deployments, through an edge-local radio-frequency identification and near-field communication door access-control case study. We (i) summarize recurring security gaps observed in typical classroom-style configurations and map them to baseline consumer-device security expectations, (ii) propose a secure-by-design reference architecture that combines unique device identities, encrypted messaging, and least-privilege topic authorization, and (iii) introduce a phase-based hardening workflow with a mandatory security gate supported by collaborative peer threat mapping. Finally, we outline framework-level extensions, including secure scaffolding, automated credential generation, and embedded checklists, that make the secure path the default while preserving prototyping velocity. The approach shows that secure operation can become the expected end state of educational prototypes without sacrificing development speed.**

Keywords-IoT security; rapid prototyping; access control; threat modeling; IoT education.

I. Introduction

Rapid prototyping is a defining feature of IoT development in university programs, industrial training, and early proof-of-concept work. Teams operate under tight time constraints, assembling systems from declarative frameworks, pre-built components, and lightweight integration tools. While this accelerates experimentation, it often sidelines security; common insecure defaults and their implications are summarized in Section III. The rapid proliferation of Internet of Things (IoT) devices has increased the demand for accessible frameworks that enable rapid development and experimentation [2][3]. IoTempower addresses this need as an educational, open-source IoT prototyping framework designed to lower the barrier to building networked embedded systems. It provides students and practitioners with preconfigured tooling for device provisioning, Message Queuing Telemetry Transport (MQTT)-based communication [4], and over-the-air updates, enabling functional prototypes to be developed with minimal setup effort [5]. As can be deduced from the references, the authors of this paper are involved in the development of IoTempower and have a vested interest in its future. IoTempower is used in hands-on teaching contexts at several European universities, where its emphasis on declarative configuration and rapid iteration supports experiential learning. While this approach yields valuable practical insights, security considerations are frequently deprioritized in typical IoTempower-based developments. The central problem is how to introduce phase-based security practices into rapid IoT prototyping workflows. The goal is to do so without impairing development velocity. Specifically, we ask:

- How can a secure-by-design architecture for access control be integrated into prototyping workflows?
- Which security mechanisms can be introduced in phases to reduce developer friction?
- How can collaborative threat-mapping support security awareness in mixed-experience student teams?
- Which framework-level extensions meaningfully reduce recurring misconfigurations?

Our objective is not to require students to implement full industrial security controls from the outset. Instead, we combine architectural guidance, incremental hardening, and structured peer review so that final prototypes use encrypted communication, verifiable identities, least-privilege authorization, and protected updates—without a last-minute hardening sprint. We address the human factor directly. Collaborative threat mapping is a structured peer-review step in which student teams exchange an architecture diagram and sanitized configuration excerpts, then apply a *Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege* (STRIDE)-lite checklist to find and remediate vulnerabilities before final assessment [6]. Peer instruction and peer assessment are well-studied mechanisms for improving learning outcomes and review quality; we adapt that logic to threat modeling in small IoT systems.

The contributions of this paper are:

- A secure-by-design reference architecture for an edge-local radio-frequency identification (RFID) access-control system implemented with IoTempower.
- A phased security model that incrementally introduces controls and enforces a security gate before final integration.

- A collaborative, STRIDE-inspired threat-mapping exercise tailored to student teams.
- A set of lightweight IoTempower extensions that provide secure defaults, automated credential scaffolding, and an embedded security checklist.

The remainder of this paper is structured as follows. Section II summarizes background and related work. Section III presents the system model and threat baseline. Section IV analyzes IoTempower's current security posture. Section V introduces the secure-by-design reference architecture. Section VI describes the phased model and framework extensions. Section VII discusses limitations, and Section VIII concludes the paper.

II. Background and Related Work

Declarative IoT prototyping frameworks, such as IoTempower [5], Tasmota [8], and ESPHome [9], lower the barrier to entry for embedded development by abstracting networking, messaging, and OTA updates behind configuration files and reusable components. In educational settings, IoTempower typically bundles a gateway (often a Raspberry Pi or a dedicated laptop in the role of an edge server) and a dedicated Wi-Fi router for the (wireless) IoT network, an MQTT broker, and integration tooling, such as Node-RED [10], while student teams configure and flash individual nodes. On the device side, teams write small node configurations, and IoTempower handles boilerplate networking, MQTT publishing/subscribing, and OTA flashing [11]. The system's control logic is usually implemented in Node-RED flows, where all devices in the system can be integrated via MQTT. A persistent tension exists between rapid prototyping and robust security. When security mechanisms — certificate management, per-device credentials, broker Access Control Lists (ACLs) — are optional or introduce friction, developers routinely postpone them until "later" and then run out of time. Industry guidance captures minimum controls that should not be optional. ETSI EN 303 645 emphasizes no universal default passwords, secure communications, and secure update processes, while NISTIR 8259A summarizes baseline capabilities around device identification, configuration, data protection, logical access control, secure updates, and security state awareness. These are broad standards, but their core ideas translate directly into practical controls for an IoT access-control prototype: unique identities, encrypted MQTT links, topic access restrictions, and protected OTA updates. We use these baseline standards as device-centric references for concrete gaps; broader frameworks like NIST CSF 2.0 [12] and the IIC Industrial Internet Security Framework [13] are referenced for context, but are intentionally out of scope for a classroom prototyping method. Threat modeling offers a practical learning lens. STRIDE is a widely used taxonomy that helps teams ask: who can spoof identities, tamper with messages, deny actions (repudiation), learn sensitive data, disrupt availability, or escalate privileges [6]. For IoT prototypes, a STRIDE-lite checklist tailored to devices, topics, and trust boundaries can be effective without requiring a full formal model [14][15].

This work addresses the resulting gap: the lack of a lightweight, pedagogical, and secure-by-design methodology tailored for resource-constrained IoT systems and rapid development environments. We propose a method that embeds security practices directly into the development workflow through framework extensions and tailored development phases, thereby mitigating common security pitfalls without sacrificing the speed and exploratory nature of IoTempower and related tools.

Prior research has explored complementary aspects of secure IoT development and developer support. For example, Corno et al. [16] study how novice developers can be guided to better recognize and address security issues in cloud–IoT systems, while Zhang et al. [17] analyze security risks arising from developer-customized device–cloud interactions. These works highlight the importance of tooling and workflow design in shaping developer security practices, but focus primarily on cloud-centric architectures or individual development decisions.

Building on these insights, our work addresses a related but distinct problem space: security practices in edge-local IoT prototyping and classroom-based development workflows. We combine (i) a concrete secure-by-design access-control architecture, (ii) a phased security model tailored to rapid IoT prototyping, (iii) collaborative peer-driven threat mapping, and (iv) framework-level extensions that encode secure defaults within a single integrated approach.

III. System Model and Baseline Threats

A. Access-Control Use Case

The case study system is a local door access-control setup suitable for a lab or office environment. An RFID reader node (ESP32) scans a badge UID and publishes it to an MQTT topic (e.g., `access/request/door1`). A Local Authorization Service (LAS), implemented as a Node-RED flow or small service on the IoTempower gateway, validates the UID against an allowlist and publishes a grant-or-deny decision. A door actuator that triggers a solenoid via a relay on another node subscribes to the decision topic and triggers unlocking for a specific door. Systems may include additional nodes, e.g. for keypad access or displaying system status, as well as a cloud connection or smartphone app that can be used to administer the system using an ACL-protected dashboard. Figure 1 shows the architecture of the system using a central gateway node that also runs the required software stack to integrate the different nodes.

B. Trust Boundaries

The gateway is treated as trusted infrastructure; nodes are exposed (an attacker may steal or tamper with them). The network boundary matters because MQTT is a message bus: if an attacker can join the Wi-Fi or reach the broker, they can potentially observe or inject events. A second boundary exists

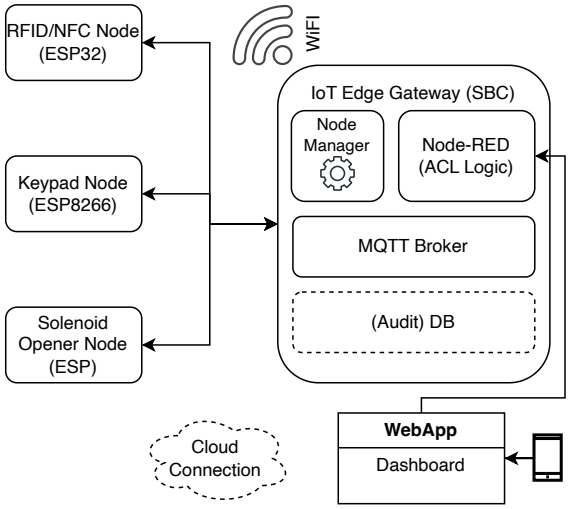

Figure 1. Access-control system architecture.

at the gateway management surface (Node-RED UI, SSH, file manager) [10].

C. Insecure Prototype Baseline

In typical early IoTempower deployments, three recurring properties are observed:

1) plaintext MQTT on port 1883 with weak or shared authentication
2) non-unique identities (all nodes share the same Wi-Fi PSK, MQTT credentials, and often the same OTA password)
3) OTA updates protected only by a password hash, with sensitive configuration embedded in firmware/flash

Table I shows a brief overview of which assets matter in this context and how they could be abused. A concrete attack scenario illustrates the impact: an attacker briefly accesses a node, dumps its flash memory, recovers Wi-Fi and MQTT credentials, joins the network, subscribes to badge and decision topics, and injects forged unlock commands. The attacker can deploy modified firmware to maintain persistence. This chain demonstrates how shared secrets and plaintext messaging collapse containment after a single device compromise.

IV. IoTempower Security Analysis

IoTempower exposes security controls, but many secrets are handled as convenience configuration. In our classroom-style IoTempower setups, a single node flash dump can often recover the Wi-Fi SSID/password, MQTT host and (optional) shared credentials, the OTA password hash (keyhash), and the MQTT/TLS CA certificate (mqtt_ca_cert). [5] As described in the insecure baseline in Section III, MQTT is frequently deployed without transport protection and strict authorization. IoTempower supports TLS and broker-side authorization controls, but they are often configured late or inconsistently in classroom-style prototyping [18]. IoTempower is secure-capable but not secure-by-default—making its gaps measurable and good targets for an incremental hardening plan. Therefore, we map the baseline prototype against key expectations from ETSI EN 303 645 and NISTIR 8259A, focusing on default credentials, secret storage, update mechanisms, transport security, and device identity. The goal is not formal compliance; it is to identify the highest-leverage gaps that a student-built access-control prototype should close first.

V. Secure-by-Design Reference Architecture

To address these weaknesses without imposing industrial-scale complexity, we propose a secure-by-design reference architecture built around three core principles: unique identity, encrypted transport, and least-privilege authorization. Our secure-by-design reference architecture addresses the baseline weaknesses by making identities, encryption, and authorization explicit design elements rather than optional configuration toggles. These elements are:

- **Unique device identity:** each node is provisioned with a unique credential (preferably an X.509 client certificate and private key). The broker authenticates clients based on these credentials, enabling per-device revocation. This breaks the "one device breach = total breach" property of shared-secret deployments.
- **TLS-only MQTT:** the broker accepts device connections only over TLS and rejects plaintext connections [19]. Mutual authentication prevents unauthorized clients on the Wi-Fi from reading or injecting MQTT messages. Even if an attacker gains Wi-Fi access, they still need valid client credentials to connect to the broker [18].
- **Least-privilege authorization:** broker-side ACLs restrict each identity's allowed topics and operations (publish vs subscribe). The LAS applies a simple policy file mapping device IDs to allowed actions. This way, a compromised reader cannot publish unlock commands, and a compromised actuator cannot subscribe to other doors' topics [18].
- **OTA and configuration hygiene:** update secrets are per-device and are not shared across the fleet. Where possible, firmware signing is introduced so that only trusted binaries are accepted. The gateway logs authentication failures and denied publishes to support auditing and to provide teachable evidence during exercises.

Topics are treated as explicit interfaces rather than "secret strings," and standardized naming conventions reinforce role separation. Access-control devices must define failure behavior. In classroom prototypes we default to fail-secure (do not unlock on missing authorization), but real deployments may require different policies for life safety and egress. These decisions should be made explicitly and tested, especially when introducing availability risks (e.g., broker outages).

VI. Phased Security Model for Rapid Prototyping

Rather than expecting teams to configure certificates and ACLs on day one, the development workflow aligns security

TABLE I. ASSETS AND ADVERSARY ACTIONS (SUMMARY).

Asset / Control Surface	Why it matters	Likely adversary actions
Node credentials (Wi-Fi, MQTT, OTA, certs)	Gate access to network, broker, and updates	Extract from flash; reuse to impersonate nodes; pivot laterally
MQTT topics (request, grant/deny, logs)	Carry badge data and unlock commands	Eavesdrop; replay; inject forged grant messages
Gateway configs (broker ACLs, LAS logic, keys)	Defines authorization and trust anchors	Abuse default access; modify allowlists/flows; steal keys
Firmware/OTA binaries	Determines device behavior and secrets in code	Upload malicious firmware; downgrade; backdoor devices
Shared Wi-Fi for all devices and gateway	Spoofing to capture all transmissions for replay attacks	Eavesdrop, replay, spoof MAC address

Figure 2. Phased security model.

upgrades with functional milestones. Early phases keep friction low, while later phases enforce security as a project requirement. The overall progression is illustrated in the phased security model timeline in Figure 2, which summarizes Phases 0–4 and marks Phase 3 as a mandatory security gate before final integration.

In Phase 0 (Bootstrap), teams bring up the gateway, MQTT broker, and initial nodes so that a minimal system is reachable. At this stage, temporary development secrets are acceptable, but the setup already enforces basic network segmentation and clear, consistent topic naming to avoid unstructured growth later.

Phase 1 (Functional prototype) focuses on achieving the end-to-end door workflow: an RFID badge read triggers an access request, the Local Authorization Service decides, and the actuator unlocks the door. During this phase, teams introduce basic hygiene measures, such as separating request and decision topics, allowing logging of access events, and removing wildcard subscriptions that would otherwise blur trust boundaries.

In Phase 2 (Identity provisioning), the system is prepared for stronger guarantees without yet forcing all traffic into TLS. Teams generate per-device certificates or credentials, precreate broker accounts, and define ACL entries so that each device identity is bound to a constrained set of topics and operations.

Phase 3 (Secure baseline) is the explicit security gate also highlighted in Figure 2: a system that "works" must now also "work securely." MQTT is switched to TLS-only operation, ACLs are enforced on the broker, and insecure fallbacks, such as plaintext listeners or shared test credentials, are removed from the configuration. Crossing this gate is a precondition for final demonstrations or grading in the intended teaching setting.

Finally, **Phase 4 (Hardening)** is an optional phase for teams that have capacity or higher assurance goals. Typical enhancements include enabling OTA firmware signing, introducing credential rotation procedures, and wiring basic monitoring or anomaly detection hooks into the system to catch suspicious behavior at runtime.

A. Collaborative Threat-Mapping

Technical controls alone are insufficient if developers do not internalize an adversarial mindset and understand how design decisions influence attack surfaces. To address this, the phased model integrates a dedicated collaborative threat-mapping exercise at the security gate in Phase 3, when the system has reached a secure baseline in terms of TLS, identities, and ACLs [20][21].

After a team has completed Phase 3, it prepares a concise review bundle that captures the security-relevant aspects of its design. This bundle typically includes an architecture diagram, a structured list of MQTT topics and their intended roles, selected configuration excerpts (e.g., broker configuration, Node-RED flows) that do not reveal private keys or secrets, and a short description of the implemented security controls. The bundles are exchanged such that each team reviews the system of another team, creating a peer-to-peer review setting that mirrors code review practices in professional development.

The STRIDE-lite checklist, which has been adapted for use in the context of IoT access control, will be used by reviewers. The adapted checklist is shown in Table II. Instead of posing open questions, the checklist prompts reviewers to examine concrete aspects, such as whether an attacker could impersonate a device or the Local Authorization Service (Spoofing), modify MQTT messages or authorization lists in transit or at the gateway (Tampering), or cause badge identifiers, credentials, or configuration data to be exposed in

TABLE II. STRIDE-LITE PROMPTS USED IN PEER THREAT MAPPING (ADAPTED FROM STRIDE [6]).

STRIDE category	Example prompts for IoT access control
Spoofing	Can a rogue client impersonate a device or the LAS to publish unlock commands?
Tampering	Can MQTT messages or allowlists be modified in transit or on the gateway?
Repudiation	Are door events logged so actions can be attributed to identities?
Information Disclosure	Do badge IDs, credentials, or configs leak over the network or in storage?
Denial of Service	What happens if Wi-Fi is jammed or the broker is flooded; does the door fail-secure?
Elevation of privilege	Can a low-privilege node publish admin topics or bypass LAS decisions?

cleartext (Information Disclosure). Additional prompts cover the system's behavior under denial-of-service conditions (e.g., Wi-Fi jamming or broker overload) and whether door events are logged in a way that supports attribution and detection of privilege escalation attempts [6][14][15].

The outcome of the exercise is a short vulnerability report that each reviewer team submits. Reports are expected to contain at least a small number of concrete findings, each accompanied by a proposed mitigation or rationale, which encourages reviewers to move beyond merely identifying issues towards actively reasoning about possible fixes. The original team is required to address the reported issues or provide a justified explanation for deviations before the final assessment, thus closing the loop between review and remediation. This process elevates threat modeling from a purely theoretical activity to a collaborative learning mechanism that reinforces the importance of security controls introduced in earlier phases.

B. IoTempower Extensions for Secure Defaults

While the phased model structures when security controls are introduced, developers still benefit from tooling support that reduces the effort of "doing the right thing." To that end, the approach proposes a set of lightweight extensions to IoTempower that encode secure defaults and automate repetitive tasks, making the secure path the default.

First, project scaffolding can ship with preconfigured secure communication templates. A default broker configuration enables TLS, disables anonymous access, and includes an example ACL file [18] that reflects the separation of roles in the access-control case study (e.g., readers, actuators, Local Authorization Service, and administrative clients). This shifts the default away from plaintext MQTT with permissive access towards a configuration that already aligns with the secure baseline described in Phase 3.

Second, credential management can be supported through an integrated command-line utility that automates the generation of key material. A single command can create a local certification authority for the classroom or lab setting, generate broker and server certificates, and provision per-device client credentials along with a mapping file that associates logical device identifiers with their respective credentials. This reduces the likelihood of teams falling back to shared secrets or ad-hoc credential schemes purely for convenience.

Third, the project repository can embed a structured security checklist and a threat-mapping worksheet that are explicitly aligned with the development phases. Tooling can be extended to remind teams at certain milestones—for example, when moving from Phase 1 to Phase 2 or approaching the Phase 3 gate—to complete the relevant checklist items and prepare review materials. This integration keeps security tasks visible within the normal development workflow, instead of relegating them to separate documentation that students may overlook.

Finally, a "classroom mode" can give instructors controlled flexibility over which protections are enforced at which time. For example, temporary insecure settings can be selectively enabled when demonstrating specific attacks or failure modes, while the default configuration remains aligned with the secure baseline and the phased progression. Together, these extensions operationalize the phased model by minimizing configuration friction, encouraging consistent application of best practices, and enabling instructors to steer the trade-off between usability and enforcement in a transparent way.

C. Preliminary Observations and Evaluation Plan

Across prior course iterations, recurring issues such as plaintext MQTT, shared credentials, and over-permissive topic access have been observed. These observations motivated introducing Phase 3 as an explicit security gate. We plan to evaluate the approach in an upcoming course iteration (approximately 20 students across five teams). Quantitative metrics could include the number and severity of vulnerabilities identified during peer review and the quality of implemented fixes; this data will be collected across multiple courses. Qualitative data will be collected through surveys and interviews focusing on perceived development friction, security awareness, and the usefulness of the framework extensions.

VII. DISCUSSION AND LIMITATIONS

The proposed approach carefully balances usability and security enforcement, particularly critical in educational settings where classroom prototypes prioritize rapid development over production-grade robustness. The phased security model enables teams to achieve functional prototypes early (Phases 0–1) before progressively layering on controls at defined milestones, culminating in the mandatory Phase 3 security gate. This structure minimizes developer friction by deferring resource-intensive tasks, such as certificate provisioning, until the system is stable, while ensuring that insecure fallbacks cannot persist into final demonstrations.

The threat coverage of the reference architecture and associated exercises focuses primarily on MQTT-related threats, such as spoofing, tampering, and lateral movement through least-privilege topic authorization. While this addresses the most frequent vulnerabilities observed in IoTempower deployments (e.g., plaintext communication and shared credentials), it does not fully mitigate denial-of-service risks (e.g., broker overload or Wi-Fi jamming), supply-chain compromises during firmware updates, or sophisticated physical attacks on exposed nodes. These gaps are intentional, given the lightweight, pedagogical scope, but they highlight opportunities for extension in future iterations.

Even in a local-only deployment, security is not guaranteed by isolation alone. Exposed management interfaces, such as the Node-RED UI, SSH access to the gateway, or an open OTA port (port 8266), can significantly expand the effective attack surface if not explicitly hardened. Teams must document their secure setup comprehensively and verify it through measurements (e.g., network traffic analysis or credential extraction attempts from flash dumps) to confirm that the intended protections are operational.

The reliance on peer-driven threat mapping in Phase 3 introduces variability depending on the experience levels within student teams. While STRIDE-lite checklists and review bundles standardize the process, uneven threat modeling quality could lead to overlooked issues in some designs. Mitigating this requires instructor calibration of reviews and iterative refinement of checklist prompts based on empirical observations from course deployments.

VIII. Conclusion and Future Work

This work translates established IoT security practices into an educational rapid-prototyping workflow that is feasible for novice teams. The key contribution is a practical integration of secure defaults, a phased security gate, and collaborative threat mapping that makes secure operation the expected end state of classroom prototypes. Future work includes empirical evaluation at scale, tighter integration of per-device credential provisioning and revocation, and support for signed firmware updates. Upstream integration of the proposed IoTempower extensions into the core framework through open-source contributions will establish secure defaults as canonical best practices across educational deployments.

References

[2] J. P. Dias, F. Couto, A. C. Paiva, and H. S. Ferreira, "A brief overview of existing tools for testing the internet-of-things", in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2018, pp. 104–109. DOI: 10.1109/ICSTW.2018.00035.

[3] R. P. Machado, U. Norbisrath, and R. Jubeh, "Iot educational framework case study: Devices as things for hands-on collaboration", *Journal of Engineering Education Transformations*, vol. 37, no. Special Issue 2, pp. 287–295, 2024. DOI: 10.16920/jeet/2024/v37is2/24045.

[4] *Mqtt version 3.1.1*, Publish/subscribe messaging protocol standard, OASIS, 2014.

[5] U. Norbisrath and IoTempower Community, *IoTempower: A framework and environment for making the internet of things (IoT) accessible for everyone.* Version 0.9.5, [retrieved: March, 2026]. [Online]. Available: https://github.com/iotempire/iotempower.

[6] Microsoft, *Stride threat modeling*, https://learn.microsoft.com/en-us/security/engineering/threat-modeling, [retrieved: March, 2026].

[8] Tasmota Project, *Tasmota firmware documentation and project site*, https://tasmota.github.io/docs/, [retrieved: March, 2026], 2025.

[9] ESPHome, *Esphome documentation*, https://esphome.io/, [retrieved: March, 2026].

[10] OpenJS Foundation, *Node-red documentation*, https://nodered.org/docs/, [retrieved: March, 2026].

[11] Arduino Project and Espressif Systems, *Arduino ota / esp8266 and esp32 ota update documentation*, https://arduino-esp8266.readthedocs.io/en/latest/ota_updates/readme.html, [retrieved: March, 2026].

[12] National Institute of Standards and Technology, "The nist cybersecurity framework (csf) 2.0", NIST, Tech. Rep. CSWP 29, 2024, [retrieved: March, 2026].

[13] *Industrial internet security framework*, version 2.0, [retrieved: March, 2026], Industrial Internet Consortium, 2023.

[14] A. Shostack, "Experiences threat modeling at microsoft", in *MODSEC@MoDELS*, 2008.

[15] L. Kohnfelder and P. Garg, *The threats to our products*, Microsoft internal threat modeling document introducing STRIDE, 1999.

[16] F. Corno, L. De Russis, and L. Mannella, "Helping novice developers harness security issues in cloud-iot systems", *Journal of Reliable Intelligent Environments*, vol. 8, no. 4, pp. 601–616, 2022. DOI: 10.1007/s40860-022-00175-4.

[17] Y. Zhang, J. Li, and D. Gu, "Rethinking the security of iot from the perspective of developer customized device-cloud interaction", in *Proceedings of the ACM/SIGAPP Symposium on Applied Computing (SAC '22)*, 2022, pp. 1070–1077. DOI: 10.1145/3477314.3508389.

[18] Eclipse Foundation, *Eclipse mosquitto: Mqtt broker documentation*, https://mosquitto.org/documentation/, [retrieved: March, 2026].

[19] E. Rescorla, "The transport layer security (tls) protocol version 1.3", IETF, Tech. Rep. RFC 8446, 2018.

[20] T. Chothia and J. de Ruiter, "Learning from others' mistakes: Penetration testing iot devices in the classroom", in *USENIX Workshop on Advances in Security Education (ASE)*, 2016.

[21] M. Hamad, A. Finkenzeller, M. Hasan, M.-O. Pahl, and S. Steinhorst, "A gamified learning approach for iot security education using capture-the-flag competitions: Architecture and insights", in *NordSec*, 2024, pp. 147–165.