

ThreatLens: An Intelligent IoT Honeypot for Cyber-Attack Monitoring and Threat Analysis Using ESP32

EN Nandishwar, Fayaz Ahamed Shaikh, Amith Shetty, Prajna, HG Shiva Prasad

Alva's Institute of Engineering and Technology, Mijar, Moodbidri, Karnataka, India

Published November 2026

Abstract

Internet of Things devices are deployed in homes, offices, and industrial settings with little regard for the security of the firmware or the network stack running on them, and this makes them a persistent target for automated scanning, credential-stuffing, and botnet-recruitment campaigns. Most defensive research in this space relies on honeypots running on general-purpose computers or cloud instances, which do not reproduce the resource constraints or network footprint of an actual embedded device. This paper presents ThreatLens, a low-cost IoT honeypot built directly on an ESP32 microcontroller that exposes a decoy router login portal, a live ultrasonic-sensor status page, a simulated Telnet shell, and a set of common-service reconnaissance traps to attract and record unauthorized access attempts. Captured events are queued on-device and forwarded over Wi-Fi to a Flask ingestion service, which stores them in a SQLite database alongside a sliding-window rule for flagging brute-force behaviour, and a React dashboard polls the same backend to give an analyst a live view of activity. The paper describes this five-part architecture — edge firmware, log transport, backend storage, analysis, and dashboard — and reports the current implementation and validation status. The system has been built and exercised with a synthetic test harness that reproduces page views, reconnaissance probes, and brute-force sequences end to end, but it has not yet been deployed on a live network, so no field attack data is reported; we describe the test setup used so far, what it does and does not cover, and the deployment work remaining before real-world evaluation.

Keywords: IoT security; honeypot; ESP32; cyber-attack monitoring; threat analysis; embedded systems

1 Introduction

The number of internet-connected embedded devices has grown far faster than the security practices used to build them. Consumer routers, IP cameras, smart plugs, and sensor nodes are routinely shipped with default credentials, unauthenticated management interfaces, or firmware that is never updated after manufacture, and attackers have taken notice: botnets such as Mirai demonstrated that hundreds of thousands of such devices can be compromised with nothing more sophisticated than a dictionary of default passwords [1]. Because these devices are numerous, heterogeneous, and often deployed without any monitoring, understanding how attackers actually probe and exploit them in the wild is difficult using traditional network security tooling alone.

Honeypots address this gap by presenting attackers with a system that looks valuable or vulnerable while doing nothing except recording the interaction [2]. Decoy systems have a long history in network defence, and the honeypot literature has since expanded specifically into the IoT space, where low- and medium-interaction honeypots emulate individual protocols or device behaviours to lure scanners and credential-guessing bots [3, 4]. Most of this work, however, runs the honeypot as software on a conventional server or cloud instance. That is a reasonable choice for studying protocol-

level attacks such as SSH brute-forcing [5], but it does not capture how an attacker's tooling behaves against something that looks and responds like an actual constrained device — a distinction that matters because attackers increasingly fingerprint targets before committing an exploit [6].

This project, ThreatLens, takes a device-level approach: the honeypot runs directly on an ESP32 microcontroller, the same class of hardware found in a large share of inexpensive IoT products. The firmware hosts a fake router login portal and a live sensor status page backed by a real HC-SR04 ultrasonic sensor, so that both the authentication surface and the "useful" functionality an attacker expects to see are present, and an extension module adds a simulated Telnet shell and a bank of reconnaissance traps on common service ports to widen the decoy's attack surface beyond HTTP. Captured events are queued on-device and forwarded to a Flask backend, which stores them in SQLite and applies a sliding-window rule to flag probable brute-force sources, while a React dashboard polls the same backend for a live operator view. The system has reached a working prototype stage: all five parts are implemented and have been exercised together with a synthetic test client, but the honeypot has not yet been deployed on a live network to collect real attacker data.

The remainder of this paper is organized as follows. The next section surveys related honeypot and IoT security research and positions ThreatLens relative to existing systems. The implementation methodology is then described, covering the edge firmware, the transport mechanism, the backend and storage layer, the dashboard, and the multi-vector extension. The results section reports the current validation status using the synthetic test harness and discusses what remains to be measured once the honeypot is deployed. The conclusion closes with a summary of contributions and planned future work.

2 Literature survey

Honeypot research is traditionally organized by interaction level. Low-interaction honeypots emulate a narrow slice of a protocol or service and are cheap to deploy and maintain but are also comparatively easy for an automated scanner to detect, while high-interaction honeypots expose a real or near-real operating system to the attacker and consequently capture much richer behavioural data at the cost of far greater deployment and maintenance effort [2, 3]. Cowrie, one of the most widely used honeypots in this space, sits at the medium-interaction point: it emulates an SSH and Telnet server closely enough to log full brute-force attempts and shell interaction without exposing a genuine host [5], and it has been used as the data source for several attack-pattern studies, including brute-force classification with naive Bayes and session clustering with k-means [7].

Within the IoT-specific honeypot literature, several systems attempt to emulate whole device behaviours rather than a single protocol. ThingPot emulates an interactive smart-home hub to capture attacks that depend on multi-step device interaction [8], and U-PoT targets the UPnP protocol used by many consumer IoT products [9]. SIPHON takes a different approach, using real physical IoT devices connected through geographically distributed network wormholes to build a scalable high-interaction honeypot without the cost of maintaining a large hardware fleet at every deployment site [10]. More recent work has explored making honeypot responses adaptive: AllPot uses a transformer-based chatbot and reinforcement learning to select responses that are more likely to keep an attacker engaged through the reconnaissance phase, on the reasoning that a longer session yields more information about the attacker's intent [6], and IoTZeroJar focuses specifically on separating zero-day exploit traffic from previously seen attack signatures [12].

A second strand of related work focuses on what a captured dataset is used for once it exists. Vishwakarma and Jain use honeypot-collected traffic as training data for a machine-learning classifier aimed at detecting IoT botnet DDoS activity [11], echoing the broader lesson from the Mirai retrospective, which found that a small number of attack patterns — mainly default-credential scanning — accounted for the overwhelming majority of device compromises [1]. Doshi et al. take a complementary angle, applying machine learning directly to network traffic from consumer IoT devices to flag DDoS activity without relying on honeypot data at all [13]. Broader surveys situate these individual systems within the wider IoT security landscape: Franco et al. provide a taxonomy of IoT, IIoT, and CPS honeypots and note that most published systems still assume a conventional-server deployment model [4], and Alzaylaee's review of smart-home device vulnerabilities highlights weak authentication and inconsistent firmware update practices as the dominant root causes of compromise [14] — both consistent with the login-portal attack surface that ThreatLens is built to observe. Closer to the hardware layer used in this project, recent ESP32-based tooling such as Slipper Zero+ demonstrates that the same class of low-cost microcontroller used here is already being used for offensive wireless security testing, which reinforces that it is a realistic stand-in for the devices attackers actually target [15].

Table 1 summarizes the systems most directly relevant to ThreatLens.

Table 1. Summary of related honeypot and IoT security literature.

Author(s), year	Focus	Contribution	Relevance to ThreatLens
Provos [2], 2004	Virtual honeypot framework	Introduced honeyd, enabling one host to simulate many virtual honeypots at different network addresses	Establishes the low-interaction baseline against which device-level honeypots like ThreatLens are compared
Razali et al. [3], 2018	IoT honeypot survey	Reviews IoT honeypot design factors and open research issues from a practitioner's viewpoint	Frames the design trade-offs (interaction level, deployment cost) ThreatLens makes explicit
Guarnizo et al. [10], 2017	Physical IoT honeypot at scale	SIPHON: routes traffic to real IoT devices via distributed wormholes	Alternative to emulation; motivates why ThreatLens instead runs directly on real embedded hardware
Wang et al. [8], 2018	Interactive IoT honeypot	ThingPot emulates a smart-home hub to capture multi-step attacks	Precedent for whole-device emulation rather than single-protocol emulation
Hakim et al. [9], 2018	UPnP-focused honeypot	U-PoT framework for UPnP-based IoT devices	Shows protocol-specific honeypots as an alternative design point to ThreatLens's HTTP and Telnet surfaces
Vishwakarma and Jain [11], 2019	ML-based botnet detection	Uses honeypot-collected data to train a DDoS-detection classifier	Suggests a future-work direction for ThreatLens's captured dataset
Antonakakis et al. [1], 2017	Botnet measurement study	Seven-month retrospective analysis of the Mirai IoT botnet	Motivates the credential-guessing threat model ThreatLens's login portal targets
Mfogo et al. [6], 2023	Adaptive honeypot interaction	AIIPot uses a transformer and reinforcement learning to extend attacker session length	Identifies session length and response realism as key metrics for ThreatLens's planned evaluation

Author(s), year	Focus	Contribution	Relevance to ThreatLens
Ellouh et al. [12], 2022	Zero-day attack detection	IoTZeroJar filters malicious traffic to separate known from zero-day attack patterns	Relevant to ThreatLens's brute-force flagging rule and future anomaly-detection work
Oosterhof [5]	SSH/Telnet honeypot	Cowrie, a widely deployed medium-interaction honeypot	Reference point for ThreatLens's own simulated Telnet shell and session logging
Doshi et al. [13], 2018	Traffic-based DDoS detection	ML classifier for IoT DDoS traffic using consumer device network features	Complementary detection approach to the log-based analysis used in ThreatLens
Franco et al. [4], 2021	IoT/IIoT/CPS honeypot survey	Taxonomy and design-factor analysis across the honeypot literature	Positions ThreatLens's ESP32-hosted design within the broader honeypot taxonomy
Alzaylaee [14], 2025	Smart-home vulnerability review	Systematic review of smart-home device security weaknesses	Corroborates weak authentication as the primary attack surface ThreatLens is built to observe
Samadram [7], 2026	Attack pattern clustering	Applies k-means clustering to Cowrie honeypot session logs	Candidate analysis technique for ThreatLens's captured login-attempt dataset
Irfan et al. [15], 2026	ESP32 security tooling	Slipper Zero+, an ESP32-based wireless penetration and defense toolkit	Confirms ESP32 as a realistic, actively used platform for both offensive and defensive IoT security work

Taken together, the literature shows a progression from generic, protocol-level honeypots toward systems that emulate specific device behaviour more convincingly, and a growing interest in what the collected data is used for afterward. ThreatLens sits at the device-emulation end of that spectrum: it runs the decoy directly on the hardware attackers target and pairs that with a physical proximity sensor, linking network-layer events to a physical signal — something none of the surveyed systems attempt.

3 Implementation methodology

ThreatLens is organized into five cooperating parts: an edge honeypot node built on an ESP32, a log-transport mechanism from that node to a backend, a Flask and SQLite backend that stores and screens events, a React dashboard that gives an analyst a live view, and a multi-vector extension that widens the decoy beyond HTTP. All five parts have been implemented and exercised together; Figure 1 shows the resulting data flow.

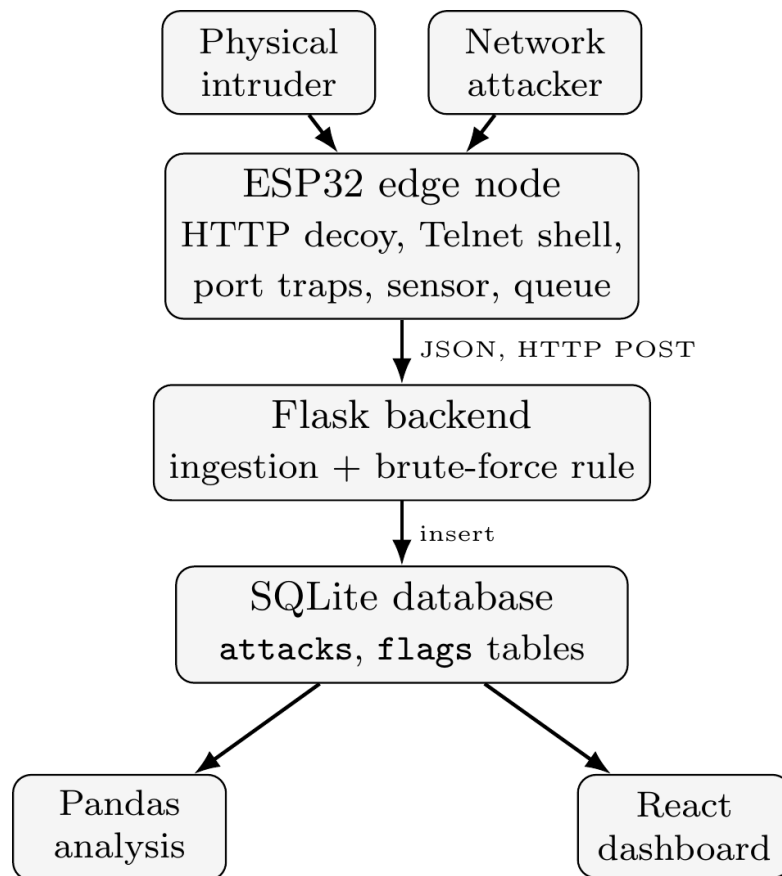


Figure 1. ThreatLens architecture: physical and network events are queued on the ESP32 edge node and dispatched to a Flask/SQLite backend, which feeds both offline analysis and a polling dashboard.

3.1 Edge honeypot firmware

The decoy runs on an ESP32 DevKit V1 using the ESP32 Arduino core and FreeRTOS. An asynchronous web server presents a login form styled after a router admin panel; every submission is recorded with the client IP, submitted username and password, and a timestamp, and the portal always returns an invalid-credentials response so automated credential-stuffing continues rather than stopping after one try. A companion status page shows a live reading from an attached HC-SR04 ultrasonic sensor, wired to the trigger and echo GPIO pins and sampled once per second, converting echo duration to distance over a 2 cm–400 cm range; this reading is folded into every logged event, linking a network interaction to the physical proximity measured at the same moment. Requests to unknown paths are logged as probes rather than silently dropped, capturing reconnaissance activity such as directory scanning.

Because the ESP32 cannot hold a large backlog of events, each one is pushed into a fixed-size circular queue guarded by a FreeRTOS mutex and drained by a background task that serializes the oldest entry to JSON and posts it to the backend; if the queue fills faster than it can be drained, the oldest entry is dropped rather than blocking the web server. Delivery uses an exponential backoff: the dispatcher waits a short interval after a failed post and doubles that wait on each further failure up to a ceiling, resetting to the short interval as soon as a post succeeds, so a temporary loss of Wi-Fi does not stall the decoy or flood the network once connectivity returns.

Figure 2 shows the breadboard prototype used for firmware development: an ESP32 DevKit V1 wired to an HC-SR04 ultrasonic sensor on a shared breadboard, matching the sensor configuration described above.

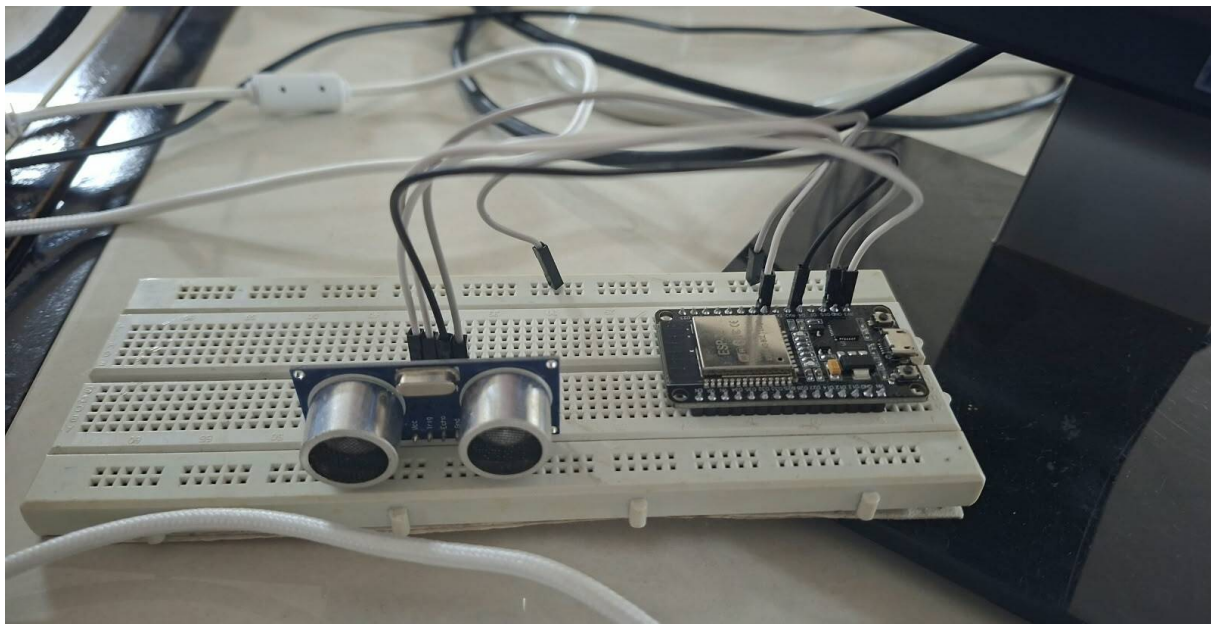


Figure 2. Breadboard prototype of the ESP32 edge node with the HC-SR04 ultrasonic sensor wired for development and bench testing.

3.2 Backend and storage

A Flask service exposes an ingestion endpoint that checks a shared key on each request, sanitizes and bounds the length of every field, and inserts the event into a SQLite database with separate tables for raw interaction records and triggered alerts, indexed on source IP, arrival time, and event type. On top of storage, the backend applies a sliding-window rule that counts login attempts from the same source IP within a fixed window and raises a flag once a configurable threshold is crossed, with a short cooldown so a single sustained attack does not flood duplicate alerts. A separate offline script reads the same database with Pandas to compute descriptive statistics such as attempt volume over time, the most common usernames and passwords attempted, and per-source connection counts.

3.3 Monitoring dashboard

A React dashboard, built with Vite and Tailwind CSS, polls the backend's summary and record endpoints every few seconds and renders the results as a set of live cards: total event volume with a threat-posture indicator, a radar-style view of the current sensor reading with distance-based alert zones, a list of active brute-force flags, rankings of the most frequently attempted credentials, and a searchable, filterable table of individual events with a detail view for inspecting a single record. Figure 3 shows a representative layout of this dashboard populated with placeholder data, since the honeypot has not yet been deployed to a network from which a real screenshot of live traffic could be taken; the components and the fields they display are unchanged from what the implemented dashboard renders once pointed at a populated database.

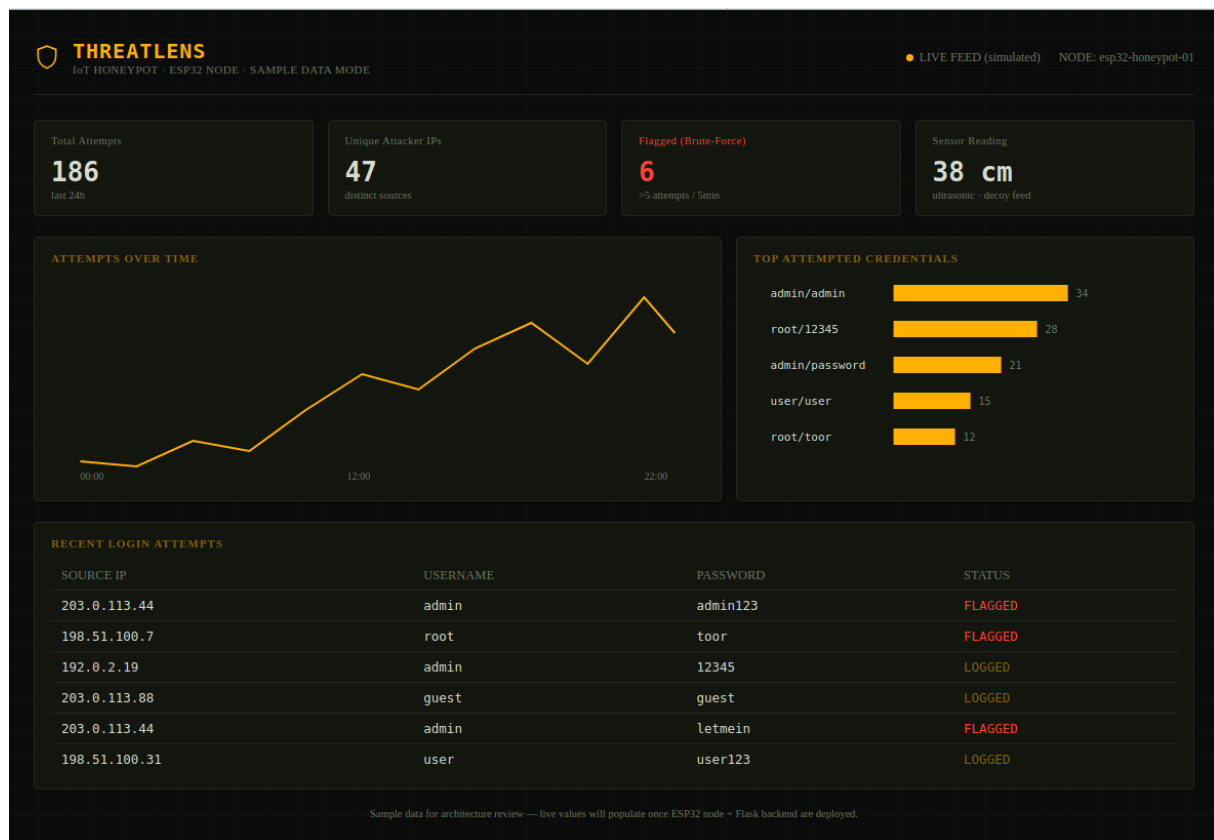


Figure 3. Representative layout of the ThreatLens dashboard, shown with placeholder data pending a live deployment to capture real traffic.

A separate admin-only panel exposes the raw sensor telemetry feed directly, reporting the transducer type, the GPIO pins it is wired to, and its rated sensing range, as shown in Figure 4. At the time of capture the ESP32 node was not connected to the network, so the panel reports the sensor as offline; the layout and the fields shown are otherwise unchanged from what the panel displays once a live reading is being sampled.

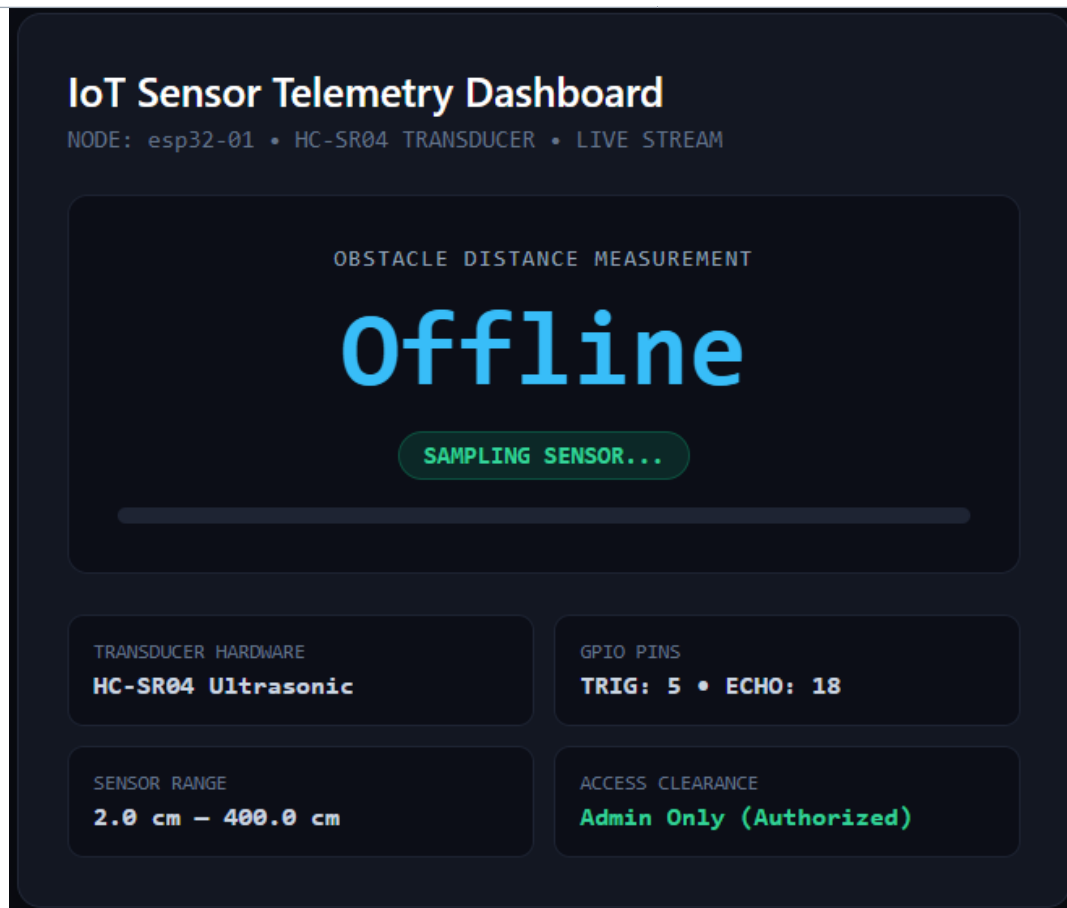


Figure 4. Admin-only sensor telemetry panel, showing the HC-SR04 configuration and sensing range; the node was offline at capture time.

3.4 Multi-vector extension

A second firmware variant extends the HTTP decoy with a simulated Telnet shell on port 23 that accepts a small number of concurrent sessions, captures submitted credentials, and emulates a limited BusyBox-style command set so an attacker past the login prompt keeps generating loggable activity. The same firmware opens short-lived listeners on eight further ports commonly associated with FTP, SSH, SMTP, POP3, SMB, RDP, VNC, and an alternate HTTP service; each connection receives a brief banner and is logged as a port-scan event before closing. This widens the set of attacker tools that register an interaction beyond those targeting a web login form.

3.5 Known implementation gap

One mismatch has not yet been corrected: the backend's brute-force rule activates only for events typed as a generic login attempt, while the live HTTP decoy logs failed logins under a more specific unauthorized-login type. A real attacker's repeated failed logins are therefore stored and visible on the dashboard but do not yet trigger an automatic flag — a gap the synthetic test client does not surface, since it sends events under the type the rule expects. The results section returns to this as a concrete item for the next revision.

4 Results and analysis

ThreatLens has reached a working prototype: all five parts described above are implemented and have been exercised together end to end using a synthetic test client that drives the full pipeline from the edge node's event queue through the Flask backend to the dashboard. The honeypot has not yet been deployed on a live network, so this section reports what the test harness has and has not validated rather than any measurement of real attacker behaviour, and no capture counts, credential statistics, or detection-accuracy figures are reported because none have been collected yet.

The test harness exercises four scenarios by sending the same JSON payloads the ESP32 would produce directly to the backend's ingestion endpoint: a single decoy page view, a probe against a nonexistent administrative path, a sequence of six rapid login attempts intended to cross the brute-force threshold, and a round of queries against the summary and record-listing endpoints. Running it confirms the ingestion endpoint accepts well-formed events, writes them with all fields intact, and that the dashboard's polling queries return the expected aggregate counts — validating the storage and presentation halves of the pipeline independently of the ESP32 hardware. It does not validate the on-device portions: queue behaviour under sustained load, backoff recovery after a genuine Wi-Fi interruption, and sensor-reading accuracy have only been exercised individually during firmware development, not end to end.

The gap identified under "Known implementation gap" is the most direct, actionable finding from this stage of testing: because the brute-force rule's trigger condition does not match the event type the live firmware actually sends, the rule as currently wired will not fire against real intruder traffic even though the underlying sliding-window count is implemented correctly, and the synthetic harness did not catch this because it was written against the rule's expected input rather than the firmware's actual output. Correcting that mismatch, and then validating the rule's true-positive and false-positive behaviour against logged attempts from a live deployment, is the immediate next step before any detection-accuracy claim can be made. Beyond that, the planned evaluation once the ESP32 is deployed on a residential or lab network will track capture completeness against the edge node's own serial-console log, the volume and composition of attempts over a multi-day window, and whether the dashboard remains responsive under the polling load of continuous traffic.

5 Conclusion

This paper presented ThreatLens, an IoT honeypot running directly on ESP32 hardware across five cooperating parts: an edge node combining an HTTP decoy, a simulated Telnet shell, and reconnaissance port traps with a physical ultrasonic sensor; a resilient on-device event queue and backoff dispatcher; a Flask and SQLite backend with a sliding-window brute-force rule; a React dashboard; and the glue connecting them. All five parts are implemented and validated together using a synthetic test harness, but the system has not yet been deployed on a live network, so this paper reports validation status rather than field results. The immediate next step is correcting the event-type mismatch identified above so the brute-force rule fires against real traffic, followed by deployment to collect a first genuine attack dataset against which capture completeness and flagging accuracy can be measured. Longer term, the project can extend along the directions raised in the literature survey: a learned classifier over the collected data [11, 13], authenticated dashboard endpoints, and a small honeynet of multiple ESP32 nodes to compare attack patterns across vantage points.

References

- [1] Antonakakis M, et al. Understanding the Mirai botnet. In: *Proc. 26th USENIX Security Symp.*, Vancouver, BC, Canada. 2017. p. 1093–1110. <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>
- [2] Provos N. A virtual honeypot framework. In: *Proc. 13th USENIX Security Symp.*, San Diego, CA, USA. 2004. <https://www.usenix.org/conference/13th-usenix-security-symposium/virtual-honeypot-framework>
- [3] Razali MF, Razali MN, Mansor FZ, Muruti G, Jamil N. IoT honeypot: A review from researcher's perspective. In: *Proc. 2018 IEEE Conf. Application, Information and Network Security (AINS)*. 2018. p. 93–98. <https://doi.org/10.1109/AINS.2018.8631494>
- [4] Franco J, Aris A, Canberk B, Uluagac AS. A survey of honeypots and honeynets for Internet of Things, industrial Internet of Things, and cyber-physical systems. *arXiv preprint arXiv:2108.02287*. 2021. <https://arxiv.org/abs/2108.02287>
- [5] Oosterhof M. *Cowrie SSH/Telnet honeypot* [Internet]. GitHub repository. <https://github.com/cowrie/cowrie>
- [6] Mfogo VS, Zemkoho A, Njilla L, Nkenlifack M, Kamhoua C. AIIPot: Adaptive intelligent-interaction honeypot for IoT devices. *arXiv preprint arXiv:2303.12367*. 2023. <https://arxiv.org/abs/2303.12367>
- [7] Samadram G. *Discovering SSH attack patterns using Cowrie honeypot and k-means clustering*. International Journal of Computer Applications. 2026.
- [8] Wang M, Santillan J, Kuipers F. ThingPot: An interactive Internet-of-Things honeypot. *arXiv preprint arXiv:1807.04114*. 2018. <https://arxiv.org/abs/1807.04114>
- [9] Hakim MA, Aksu H, Uluagac AS, Akkaya K. U-PoT: A honeypot framework for UPnP-based IoT devices. In: *Proc. 2018 IEEE 37th Int. Performance Computing and Communications Conf. (IPCCC)*. 2018. p. 1–8. <https://doi.org/10.1109/IPCCC.2018.8711321>
- [10] Guarnizo JD, Tambe A, Bhunia SS, Ochoa M, Tippenhauer NO, Shabtai A, et al. SIPHON: Towards scalable high-interaction physical honeypots. In: *Proc. 3rd ACM Workshop on Cyber-Physical System Security*. 2017. p. 57–68.
- [11] Vishwakarma R, Jain AK. A honeypot with machine learning based detection framework for defending IoT based botnet DDoS attacks. In: *Proc. 2019 3rd Int. Conf. Trends in Electronics and Informatics (ICOEI)*, Tirunelveli, India. 2019. p. 1019–1024. <https://doi.org/10.1109/ICOEI.2019.8862720>
- [12] Ellouh M, Ghaleb M, Felemban M. IoTZeroJar: Towards a honeypot architecture for detection of zero-day attacks in IoT. In: *Proc. 2022 14th Int. Conf. Computational Intelligence and Communication Networks (CICN)*, Al-Khobar, Saudi Arabia. 2022. p. 765–771.
- [13] Doshi R, Apthorpe N, Feamster N. Machine learning DDoS detection for consumer Internet of Things devices. *arXiv preprint arXiv:1804.04159*. 2018. <https://arxiv.org/abs/1804.04159>
- [14] Alzaylaee MK. A systematic review of security vulnerabilities in smart home devices and mitigation techniques. *arXiv preprint arXiv:2507.01018*. 2025. <https://arxiv.org/abs/2507.01018>
- [15] Irfan RS, et al. *Slipper Zero+: ESP32-based Wi-Fi penetration and defense toolkit*. International Journal of Engineering Research & Technology (IJERT). 2026.