أكاديمية كاوست
KAUST ACADEMY

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

BASEERA:
Threat Intelligence Aggregator from Distributed Honeypots
Final Program Report

Abdulaziz Alharbi, Abdullah Milibary, Hussam Alsumih,
Khalid Almughamsi, Mohsen Alsayed, Prof. Insaf Ullah Khattak

aboshxm2@gmail.com, dblatsh@gmail.com, qihossam1@gmail.com,
khaled21900@gmail.com, mhsnalsayed@gmail.com, insaf.ullah@essex.ac.uk

Abstract

BASEERA (Arabic for insight, or clear perception) is a distributed honeypot platform that collects attack data from multiple sensor nodes and aggregates it into a centralised threat-intelligence system. Across four weeks, a five-person team took the idea from a single Cowrie honeypot running on one laptop to a three-region sensor fleet feeding a live collector, an enrichment pipeline that produces attacker profiles, IP reputations, and geolocation insight, an alerting engine built on seven detection rules, and a seven-section web dashboard for real-time monitoring and analytics. Aggregated intelligence is exported as actionable feeds (JSON, CSV, and STIX 2.1) for downstream security teams. The finished system was presented to the program's sponsors on August 11, 2026, to a successful reception, with positive feedback from both the sponsors and the mentor. Since then, the team has carried the project past the program's own scope, reaching a stretch goal set out in its very first design document: *dionaea*, a second, differently-shaped honeypot that impersonates vulnerable Windows network services and captures the malware attackers try to plant rather than just the commands they type. This report tells that whole story: architecture, results, and honest limitations, as a single account of what BASEERA is and how it was built.

Date: August 23, 2026

Contents

1 Introduction **2**

2 Related Work **3**

3 Methodology **3**

3.1 System overview . . . 3

3.2 Sensors: Cowrie and dionaea (Hussam) . . . 4

3.3 Central collector (Mohsen) . . . 4

3.4 Threat-intelligence enrichment (Khalid) . . . 5

3.5 Database, alerting, and export (Abdulaziz) . . . 5

3.6 Dashboard (Abdullah) . . . 5

3.7 Engineering practices . . . 5

4 Results **5**

5 Discussion **10**

6 Limitations and Future Work **10**

7 Conclusion **11**

References **12**

1 Introduction

Honeypots are decoy systems designed to look like genuine, vulnerable services so that they attract and record unauthorised access attempts. Because every connection to a honeypot is, by definition, unsolicited, what it captures is a low-noise source of real attacker behaviour: credentials being tried, tools being staged, commands being run once "inside". A single honeypot only sees the attackers that happen to reach it, though, so aggregating many honeypots deployed across different networks is what turns isolated sightings into actionable threat intelligence.

Project description. BASEERA builds a distributed honeypot platform that collects attack data from multiple nodes and aggregates it into a centralised threat intelligence system. The system simulates various vulnerable services, logs attacker interactions, and processes the data to generate attacker profiles, IP reputations, and geolocation insights. Aggregated threat information can be exported as actionable feeds for defensive security teams, and the platform provides a web-based dashboard for real-time monitoring, analytics, and alerting.

The team. Five students built BASEERA as their capstone for the KAUST Academy Cybersecurity Specialization, under mentor Prof. Insaf Ullah Khattak: Hussam Alsumih (sensor infrastructure), Mohsen Alsayed (central collector), Khalid Almughamsi (threat-intelligence enrichment), Abdulaziz Alharbi (database and alerting), and Abdullah Milibary (dashboard). Section 3 credits each member's part in detail.

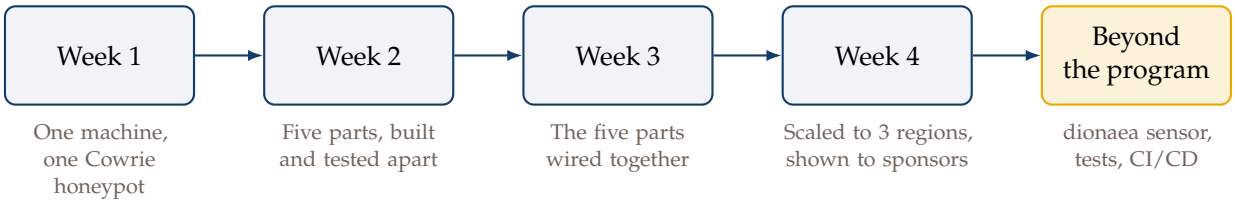


Figure 1 BASEERA's four-week program arc, and what followed it.

How the project got here. Figure 1 summarises the shape of the work. Week 1 validated a single-machine pipeline: one Cowrie honeypot, one local database, one dashboard, run and verified by every member independently. Week 2 split that pipeline into five owned parts (sensors, collector, database, enrichment, dashboard), built and tested in parallel, mostly against dummy data. Week 3 wired those five parts into one live system and proved it with a real, if small, end-to-end attack chain. Week 4 scaled the sensor fleet to three AWS regions, matured the alerting and export pipeline, rebuilt the dashboard into a seven-section product, and delivered real operating numbers for the first time. That system, under the working name *Threat Intelligence Aggregator from Distributed Honeypots*, later renamed **BASEERA**, was presented to the program's sponsors on August 11, 2026, as the team's Cybersecurity Specialization capstone. The presentation was a success, with positive feedback from both the sponsors and the mentor.

The team kept building after the program's formal end. The project's own frozen design contract had named a second honeypot, dionaea, and STIX/TAXII export as *stretch goals*, reached only if time allowed. STIX export shipped in Week 3, and dionaea is this report's headline addition, described in Section 3 and discussed as the project's main achievement beyond the program itself in Section 5.

Section 2 places BASEERA against the tools it is built on, Section 3 walks through the finished architecture part by part, Section 4 presents it in operation, Section 5 reflects on the whole arc, and Section 6 closes with the project's own honestly documented rough edges and what a next phase should address.

2 Related Work

BASEERA is built on established open-source tools rather than reimplementing them. Two honeypots answer different questions: Cowrie [1] is an SSH/Telnet shell emulator that records what an attacker *typed* once logged in, while dionaea [2] impersonates vulnerable Windows network services and records what an attacker *sent*, up to and including malware they tried to plant. The central collector is a FastAPI [3] service, and the dashboard is a Flask [4] application reading the shared database read-only. Enrichment combines IP geolocation from ip-api.com [5] with reputation scoring from AbuseIPDB [6]. Aggregated intelligence is published as STIX 2.1 [7] indicator bundles, structured for any consumer that speaks TAXII [8], the companion exchange protocol OASIS defines alongside STIX.

As in every progress report through the program, this section stays narrow rather than padded: it names what BASEERA is actually built on, not a survey of the field. A proper literature review, covering prior distributed-honeypot architectures, comparison with commercial threat-intelligence platforms, and known weaknesses of low/medium interaction honeypots such as attacker fingerprinting, remains useful future work, noted again in Section 6.

3 Methodology

3.1 System overview

Figure 2 shows BASEERA as it stands: sensor hosts run honeypots and a shared shipping library that speaks one contract to a central aggregator host, which runs the collector, the alert engine, and the enrichment worker against one SQLite database, read by a separate dashboard process. Every event, from either honeypot, passes through the same envelope, **Baseline v1.3**, so the collector, the alert rules, and the dashboard cannot tell which sensor produced a row except by its `protocol` field. That single shared contract, frozen early and amended only additively, is what let five people build five parts in parallel in Week 2 and have them actually fit together in Week 3.

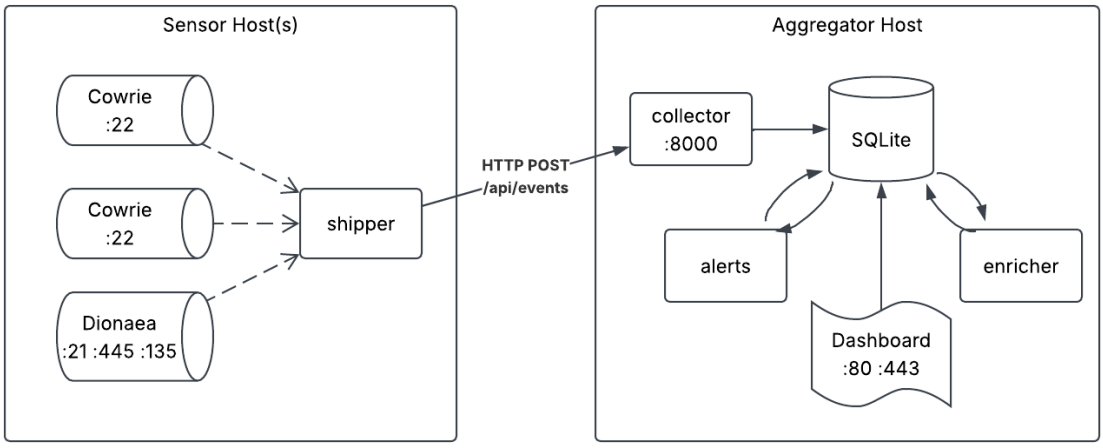


Figure 2 BASEERA's architecture: sensor hosts ship events over HTTP to a collector, which feeds a shared SQLite database read by the alert engine, the enrichment worker, and the dashboard.

3.2 Sensors: Cowrie and dionaea (Hussam)

Honeypot nodes are the only part of BASEERA meant to run on separate machines. By Week 4 they ran across three AWS regions (`us-west-1`, `eu-north-1`, `ap-northeast-1`). Two honeypot types now feed the same pipeline. **Cowrie** emulates an SSH/Telnet shell and answers "what did the attacker type once they were in." **dionaea** [2], added as this report's headline extension, impersonates vulnerable Windows network services and answers a different question: "what did the attacker *send*", up to and including the malware payload itself. Where BASEERA's original frozen design named only two `protocol` values for a second sensor (`ftp`, `smb`), dionaea speaks fourteen. Amendment 1 to the shared schema extended the contract to name the rest, additively, so no sensor already shipping had to change (Table 1).

Table 1 dionaea services mapped onto BASEERA's shared `protocol` contract.

dionaea service	shipped as	dionaea service	shipped as
`ftpd`	`ftp`	`TftpServerHandler`	`tftp`
`smbd`	`smb`	`upnpd`	`upnp`
`httpd`	`http`	`mqttd`	`mqtt`
`mysqld`	`mysql`	`Memcache`	`memcache`
`mssqld`	`mssql`	`mongod`	`mongo`
`SipSession`	`sip`	`printerd`	`printer`
`epmapper`	`epmap`	`pptpd`	`pptp`

Not everything dionaea sees is shipped, and the choices are deliberate rather than an oversight. Connections to `blackhole` (dionaea's sink for ports with nothing behind them) are counted but dropped, since a scan hit is not an attributable protocol. The second channel of an FTP transfer is folded into the session already reported through `ftpd` rather than split in two. Outbound connections dionaea itself makes to fetch a payload an exploit pointed it at are excluded entirely, because the remote end there is a malware host, not the attacker, so recording it as `attacker_ip` would put the wrong address in an exported threat feed. The adapter reports what it skipped on a timer rather than staying silent about it.

Both sensors reuse the same underlying *shipper* library rather than duplicating shipping logic: it tails the honeypot's JSON log across log rotation, batches events, sends a heartbeat every 60 seconds so the collector can tell whether a node is still alive, and spools failed batches to disk for automatic retry rather than losing them. Only each honeypot's own event-to-envelope mapping differs between the two sensor types.

3.3 Central collector (Mohsen)

The collector is a FastAPI [3] service exposing a single `POST /api/events` endpoint. Every request carries `X-Node-ID`/`X-Node-Key` headers checked against a configured identity per node, so a node can only submit events under its own name. Each event's ID is used as a deduplication key, so a retried batch cannot be stored twice. In the finished system the collector, the enrichment worker, and the alert/export cycle run as three loops inside one long-lived aggregator process, started with a single command, a simplification the team converged on once all three needed to agree on one database.

3.4 Threat-intelligence enrichment (Khalid)

The enrichment worker turns a bare attacker IP into context: country, city, and coordinates from ip-api.com [5], an abuse score from AbuseIPDB [6] where a key is configured, and a *profile score* the team computes itself from that IP's own session and command history on the platform, so a meaningful risk signal still exists even with no external key configured at all. New attacker IPs are enriched automatically as they arrive rather than on a fixed schedule, keeping the dashboard's reputation data close to real time.

3.5 Database, alerting, and export (Abdulaziz)

Every part of BASEERA reads and writes against **Baseline v1.3**, the shared SQLite schema and JSON envelope frozen early in the project and extended only additively since. An alert engine re-evaluates seven detection rules on a rolling window, `brute_force`, `credential_spray`, `high_risk_ip`, `suspicious_command`, `malware_staging`, `multi_node_scan`, and `post_auth_activity`, and writes deduplicated alerts rather than re-alerting on the same condition every cycle. The same cycle exports the current state of the database as a threat feed in three formats: native JSON, flat CSV (indicators and alerts), and a STIX 2.1 [7] indicator bundle for anything that speaks TAXII [8]. The same exporter code produces both the downloadable files and the dashboard's in-browser preview, so the two cannot silently disagree.

3.6 Dashboard (Abdullah)

The dashboard is a Flask [4] application that opens the database read-only. Acknowledging or closing an alert is its only write path, and that can be disabled entirely by configuration. It grew from a handful of charts into seven sections: *Overview* (fleet-wide posture and a real-time global attack map), *Attackers* (a searchable, filterable table of every observed IP, each opening onto its own profile page of reputation, per-sensor history, and a 14-day activity chart), *Sessions* (full per-session transcripts), *Map*, *Alerts*, *Nodes*, and *Feeds* (the exportable threat intelligence). A demo-data seeding tool can populate a full day of realistic traffic with no sensors or collector running at all, useful for demonstrations. The alerts it shows are produced by the real rules engine running over that generated data, not written directly.

3.7 Engineering practices

Beyond the five parts themselves, the finished codebase reflects practices the team adopted as it matured: an automated test suite covering the collector's HTTP handling, authentication, deduplication, and contract validation, environment-based configuration (`.env` / `.env.secrets`) shared by every process rather than hard-coded settings, Docker Compose for both the aggregator and the dashboard, and a continuous-integration workflow that publishes both as container images on every commit, so a deployment host needs no source checkout at all.

4 Results

The numbers and screens below are from the finished system in continuous live operation, captured roughly two weeks after the sponsor presentation. This is the fullest snapshot of BASEERA available at the time of writing, taken from the Overview page (Figure 3):

- **306** attack events in the last 24 hours (180 in the last hour alone), against a lifetime total of **17,374** sessions, up from 9,386 at the time of the sponsor presentation: evidence the platform has kept running and collecting unattended in the weeks since.

- **362** unique attacker IPs, of which **338** (93.4%) were automatically enriched with country, city, and a risk score.
- **172** open alerts, **83** of them high severity and flagged as needing a decision.
- **3 of 3** sensors online across all three AWS regions, with 15 attacker origins currently placed on the live map.

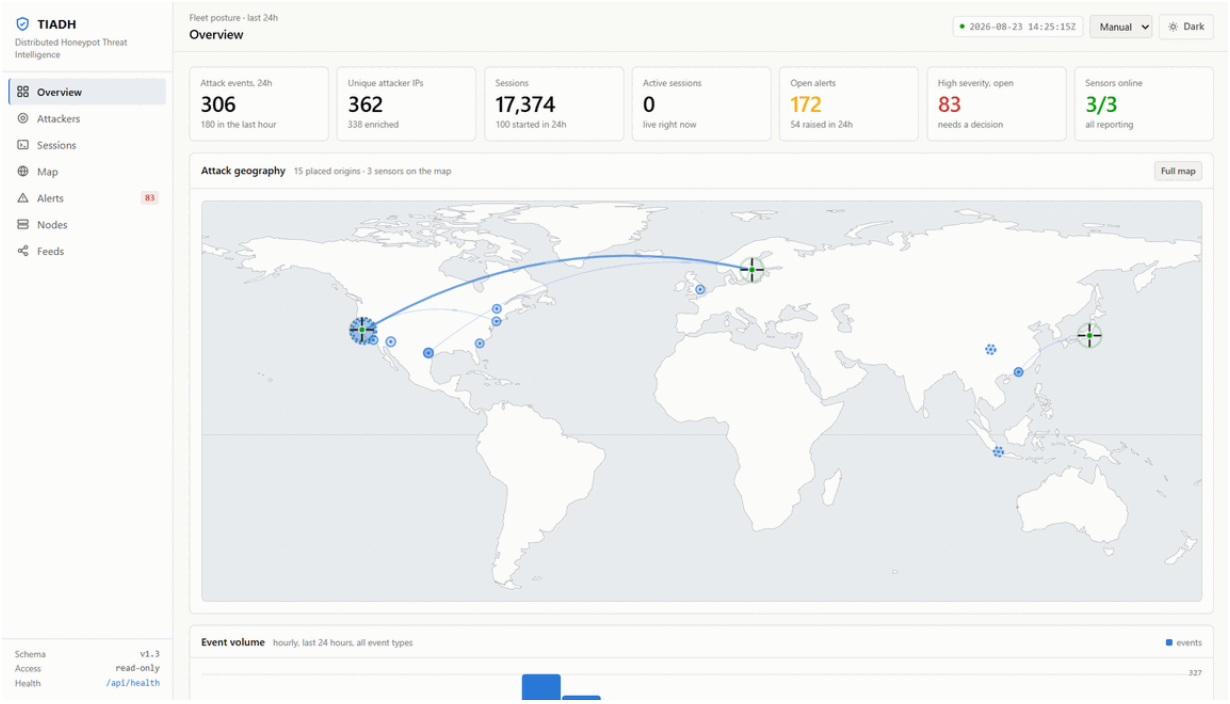


Figure 3 Overview: fleet-wide posture and a real-time global attack map, in the dashboard's light theme.

From aggregate numbers to one attacker.. The Attackers page turns those totals into a searchable, scored table of every IP the platform has seen, filterable by country, node, minimum score, alert status, or login status. Figure 4 shows it filtered to attackers from China alone, 58 of the 362 total, with per-IP columns for sessions, events, failed logins, commands, downloads, and alerts: the difference between knowing *how much* is happening and knowing *who* is doing it.

One IP, drilled all the way down.. Clicking an attacker opens a dedicated profile (Figure 5): every sensor it has touched, its enrichment record, and an activity chart over the last 14 days. The example shown, `101.201.104.216`, is a useful edge case: 238 events and 119 sessions across all three nodes, but zero login attempts and zero commands, so it never reached a shell. AbuseIPDB has no score for it, yet the platform's own *local profile score* still rates it 25, computed purely from the shape of its behaviour (broad, repeated, multi-node scanning) with no external reputation data required. This is the enrichment design choice from Section 3 working as intended: a meaningful signal exists even for an IP no third-party feed has an opinion on.

A repeat intrusion, not a one-off.. The Sessions page shows individual sessions in full. Figure 6 is the seventh of eight sessions on record from attacker `31.94.22.62`, the same IP behind the malware-download intrusion first captured at the sponsor presentation, this time against `node-02` rather than `node-03`, and running the same playbook: a failed login followed by a successful one, host reconnaissance, account manipulation (`cat /etc/passwd`), a malware download via `wget` with its SHA-256 recorded, the payload made executable and run, and `history` cleared before

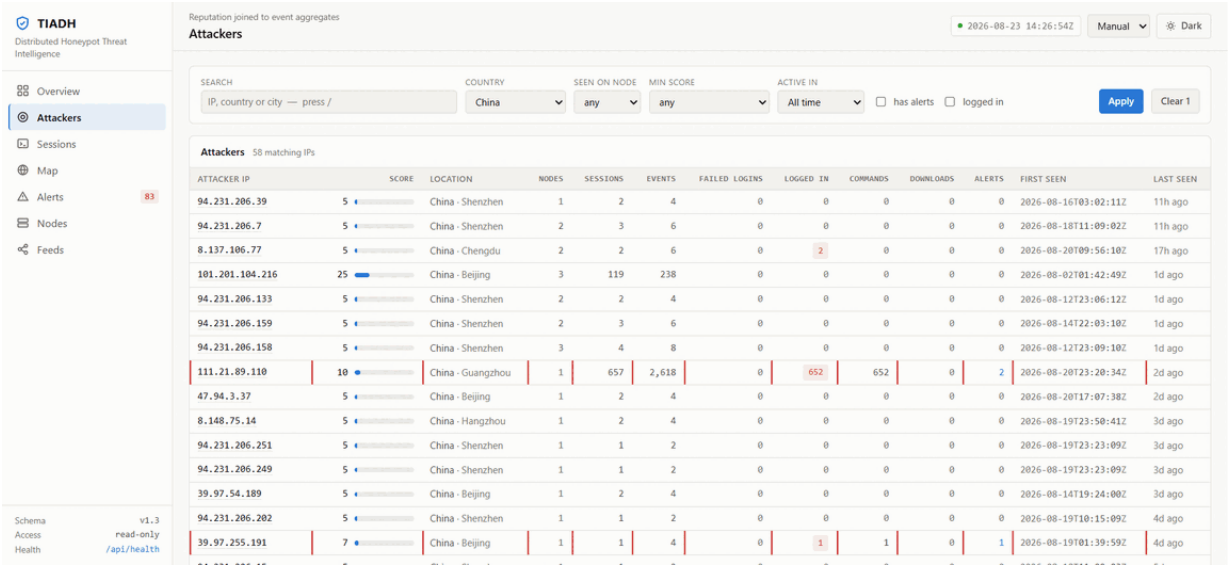


Figure 4 Attackers, filtered to China: 58 matching IPs, each broken down by sessions, events, and alerts.

disconnecting. Seeing the same attacker, the same script, repeated across different sensors over time is a stronger result than any single session: it is evidence the platform is tracking a persistent campaign, not just logging isolated events.

Exportable intelligence.. The same attacker IP from that session appears on the Feeds page as a published indicator tagged `malware_staging, multi_node_scan` (Figure 7): the same IP, the same behaviour classification, agreeing across independently rendered views, as in the sponsor-presentation snapshot. At the time captured, 13 attacker IPs were published as indicators and 90 alerts included as supporting evidence, all downloadable as JSON, CSV, or a STIX 2.1 bundle.

dionaea.. Every session shown above is still SSH, from the Cowrie side of the fleet. These screens do not yet show dionaea traffic in the numbers. Table 1 in Section 3 documents the fourteen network services dionaea now exposes across the same three regions, integrated after the sponsor presentation. Because it speaks the same Baseline v1.3 envelope as Cowrie, no query, rule, or dashboard screen needed to change to accommodate it. A dionaea session will simply appear alongside Cowrie's the next time one of its services is hit, distinguishable only by a `protocol` value like `smb` or `ftp` instead of `ssh`. That a second, structurally different honeypot required zero changes downstream of the collector is itself a result: it is the shared-contract design from Week 2 doing exactly the job it was built for.

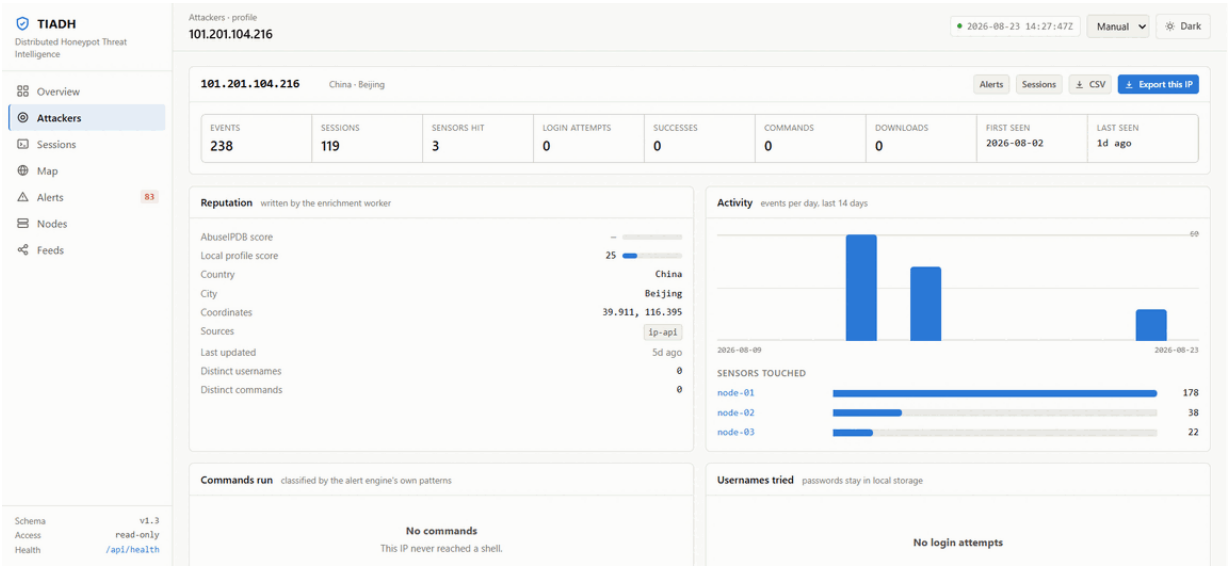


Figure 5 An attacker profile: reputation, 14-day activity, and per-sensor history for one IP, here a scanner with no logins and a purely behaviour-driven risk score.

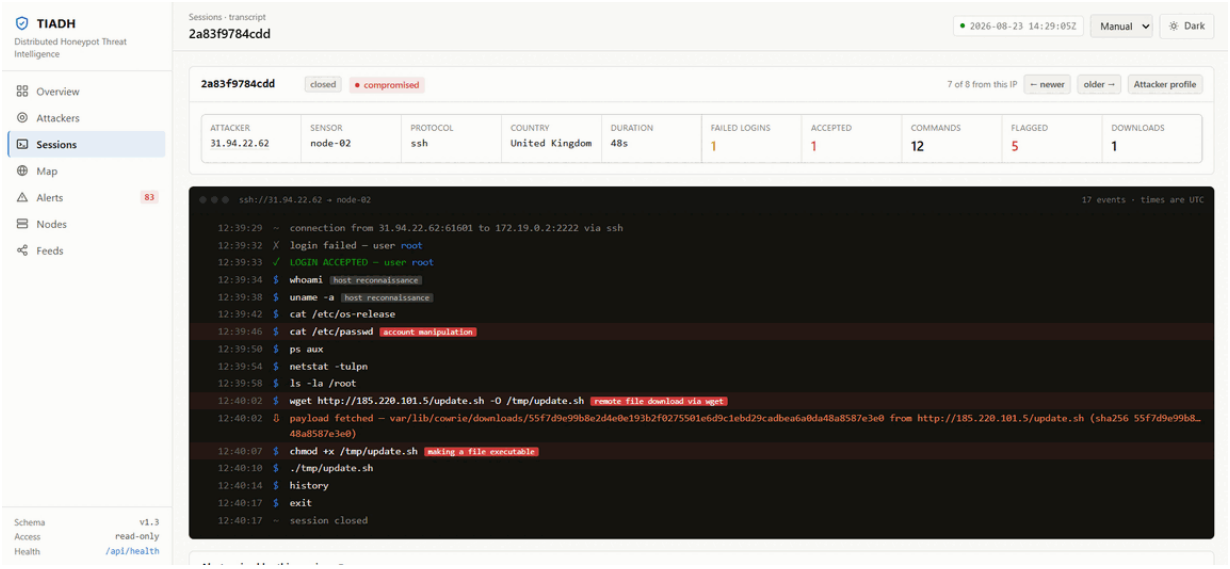


Figure 6 Sessions: the same attacker's malware-staging playbook, replayed against a different node, tracked as session 7 of 8 from this IP.

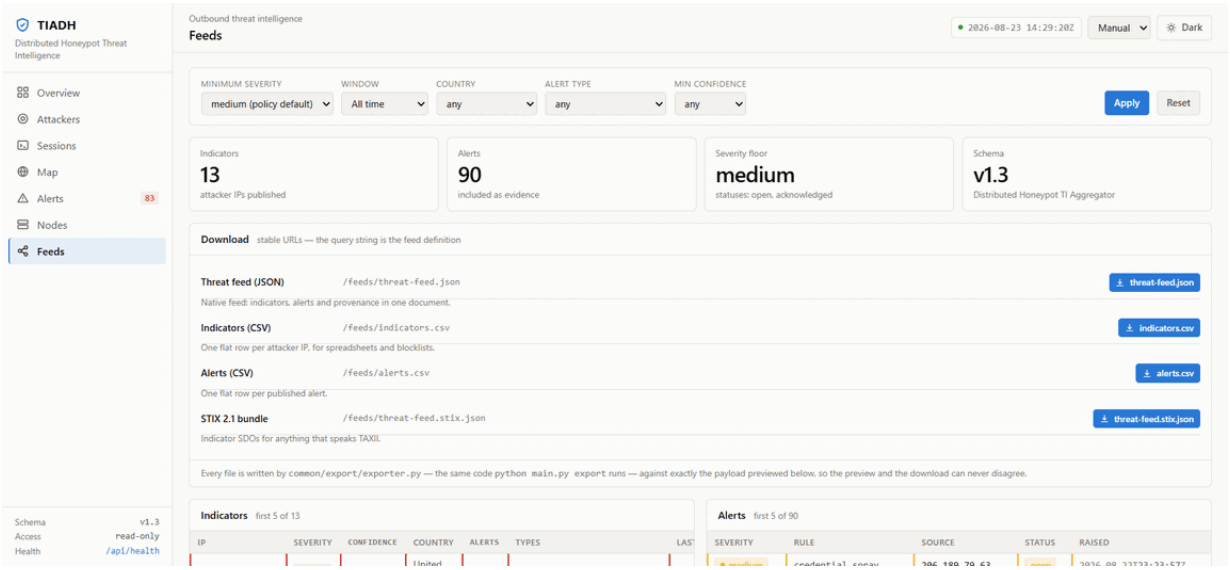


Figure 7 Feeds: 13 published indicators and 90 alerts, exportable as JSON, CSV, or STIX 2.1.

5 Discussion

Read as one story rather than four separate reports, BASEERA followed a deliberate order: prove the idea on one machine, split the work so five people could build in parallel, prove the parts actually talk to each other, and only then scale and polish. That order mattered. Each progress report's Limitations section became, in effect, the next report's to-do list. Week 1 flagged no multi-sensor deployment, and Week 2 delivered three regions. Week 2 flagged two database schemas that did not match, and by Week 3 the data flow showed one had become the system's actual source of truth. Week 3 flagged unproven alerting, and Week 4 showed the rules firing seven times on one real intrusion. Every week flagged the absence of real quantitative results, and Week 4 finally delivered them. Few student projects close every open item they raise against themselves. BASEERA closed most of them.

The clearest evidence that the underlying design was sound is what happened *after* the program ended. BASEERA's own frozen baseline document, written in the project's early days, named a second honeypot and STIX/TAXII export as stretch goals, explicitly scoped as "if time allows", not committed work. Both were reached: STIX export shipped during the program, and dionaea shipped after it, integrated with zero changes to the collector, the alert rules, or any dashboard screen. That is the payoff of freezing one shared event contract in Week 2 and treating later changes to it as strictly additive amendments rather than rewrites: a structurally different honeypot, watching completely different network services, slotted in as just another value of one field.

The project's own name is part of this story too. What began as the *Threat Intelligence Aggregator from Distributed Honeypots* was renamed **BASEERA** (insight, or clear perception) once the system had grown from a descriptive mouthful into something that actually earned a name of its own: a platform that takes noisy, distributed attack traffic and turns it into something a defender can see clearly.

None of this should read as a claim that BASEERA is finished. Section 6 is deliberately built from the project's own documented rough edges rather than a generic list, because a team that writes down its own known issues in its README, unprompted and in production-style detail, is demonstrating the kind of engineering maturity that matters as much as any single feature.

6 Limitations and Future Work

The team documents its own operational rough edges directly in the project's README, in more precise detail than an outside report could reconstruct. They are reproduced here, credited to that documentation, alongside the broader architectural points raised across the program.

- **Configuration footguns, documented by the team itself.** The adapter's default log path is relative, so a sensor started from the wrong working directory finds nothing. And `NODE_ID` defaults to a real node identifier rather than failing outright, so two same-type sensors started without an explicit ID will silently both claim to be the same node. Both are called out in the repository's own "Known rough edges" section, and both have a stated fix: set an absolute log path, and always set `NODE_ID` explicitly.
- **Enrichment depends on external, rate-limited services.** Abuse scoring needs an AbuseIPDB key to populate at all (geolocation and the platform's own profile score still work without one). ip-api.com is capped at 45 lookups/minute and AbuseIPDB at 1000/day, so a burst of new attacker IPs can outrun the enrichment worker, and a failed lookup is retried only after seven days. *Recommendation:* this is a reasonable default for a student project's traffic volume,

but a production deployment would need a paid tier or a self-hosted GeoIP database to avoid gaps.

- **One known-failing test.** The collector's test suite disagrees with its own validation code about whether a malformed event should be rejected with HTTP 200 (`rejected: 1`) or HTTP 422, documented, not silently broken. *Recommendation:* resolve which layer owns contract validation and update whichever side is wrong.
- **The aggregator and dashboard are not themselves distributed.** Only the sensors are meant to run on separate hosts. The collector, database, alerting, enrichment, and dashboard share one SQLite file on a single aggregator host by design, and the dashboard reads that file directly rather than through the collector's API. This is a deliberate, reasonable simplification for the program's scope, but it is a coupling and a scaling ceiling whoever continues the project should be aware of. Moving to a networked database and routing dashboard reads through an API would be the natural next step if sensor count or traffic grew substantially.
- **Alerting is proven but not yet measured.** Week 4 and this report both show the alert rules correctly firing on real intrusions, but no one has yet measured false-positive or false-negative rates across the full alert volume. *Recommendation:* manually review a sample of high-severity alerts to estimate precision now that real alert volume exists to sample from.
- **No formal literature review.** As every progress report from Week 1 onward has noted, BASEERA has not been formally situated against prior distributed-honeypot research or commercial threat-intelligence platforms. *Recommendation:* complete that review if the project is written up more formally beyond this program.
- **Proposed next research direction.** Beyond engineering hardening, the team has discussed extending BASEERA into an original research question: replacing Cowrie's static shell on one sensor with an LLM-backed shell on others, to study whether more convincing deception keeps an attacker engaged for longer, and whether that gain holds up against the extra response latency an LLM-backed shell introduces as a potential detection signal for the attacker. This is a proposed direction, not yet implemented, and is noted here as the project's likely next chapter rather than a completed result.

7 Conclusion

Over four weeks and beyond, five students turned a single Cowrie container on one laptop into BASEERA: a three-region distributed honeypot platform with a live collector, automatic geolocation and reputation enrichment, seven-rule alerting proven against real intrusions, a seven-section monitoring dashboard, and standards-based threat-feed export. The system was presented to the program's sponsors on August 11, 2026, to a successful reception, with positive feedback from both the sponsors and the mentor. Since then, the team has reached a stretch goal set out in the project's own earliest design document (a second, structurally different honeypot in dionaea), integrated without changing a line of the collector, the alert rules, or the dashboard, because the shared event contract the team froze in Week 2 was built to make exactly that possible. The gaps that remain are the kind a mature project takes on next, not signs the effort fell short: by any reasonable measure, a team that set out to build a distributed threat-intelligence aggregator from scratch, and finished with a working, presented, and still-growing product, met its goal.

References

[1] Michel Oosterhof and Cowrie contributors. *Cowrie SSH/Telnet Honeypot*. https://github.com/cowrie/cowrie. Accessed: 2026-07-13.

[2] DinoTools and dionaea contributors. *dionaea – catches bugs*. https://github.com/DinoTools/dionaea. Accessed: 2026-08-20.

[3] *FastAPI Documentation*. https://fastapi.tiangolo.com. Accessed: 2026-07-23.

[4] *Flask Documentation*. https://flask.palletsprojects.com. Accessed: 2026-08-20.

[5] *ip-api.com – IP Geolocation API*. https://ip-api.com/docs. Accessed: 2026-08-20.

[6] *AbuseIPDB – IP Address Abuse Reports*. https://www.abuseipdb.com/. Accessed: 2026-08-20.

[7] OASIS Cyber Threat Intelligence Technical Committee. *STIX – Structured Threat Information Expression*. https://oasis-open.github.io/cti-documentation/. Accessed: 2026-08-07.

[8] OASIS Cyber Threat Intelligence Technical Committee. *TAXII – Trusted Automated Exchange of Intelligence Information*. https://oasis-open.github.io/cti-documentation/. Accessed: 2026-08-13.