Trappyfi: An Intelligent Honeypot-as-a-service

[1]Prof. Swapnil Wani, [2]Anish Kharat, [3]Anannya Manwar, [4]Aditya Chaskar, [5]Vedant Patil

[1]Guide, [2,3,4,5]UG Scholar, Department of Computer Science and Engineering (IoT & CSIBCT) SIES Graduate School of Technology Nerul, Navi Mumbai, India.

[1]swapnilw@sies.edu.in, [2]anishpkiot122@gst.sies.edu.in, [3]anannyapmiot122@gst.sies.edu.in,

[4]adityaciot122@gst.sies.edu.in, [5]vedantspiot122@gst.sies.edu.in

***Abstract*—Modern cyber attacks are becoming more complex and harder to predict. Attackers no longer rely on a single method and often target multiple services at the same time. To study and detect such behaviour, we developed Trappyfi, an intelligent honeypot platform that simulates multiple real-world services in one system.**

Instead of deploying a single decoy, Trappyfi runs several services together, including a web login portal, SSH access, a MongoDB instance, an SMTP server, and a fake AWS console. This allows the system to capture a wider range of attack activities and present them in a unified dashboard.

To improve detection,we added a machine learning layer using the Isolation Forest algorithm, which helps identify unusual behaviour that traditional rule-based methods might miss. In addition, all captured events are securely recorded on the Ethereum Sepolia network to ensure the data cannot be altered. During a 30-day deployment, the system collected over 1,000 attack events from multiple countries and successfully identified suspicious activities in real time. These results show that Trappyfi can provide a more complete and reliable view of modern cyber

threats.

***Index Terms*—honeypot, anomaly detection, machine learning, blockchain, Isolation Forest, Ethereum, attack detection, intru- sion deception, SSH honeypot, MongoDB, SMTP, geolocation, real-time alerting**

I. Introduction

Today's cyber attacks rarely follow a single path. An attacker might begin with a simple login attempt, move to trying SSH passwords, and then probe for open databases-all within a short time. Because these actions occur in different services, many traditional security tools do not to see the full picture.

Most existing solutions focus on one single type of attack or service. This makes it difficult to understand how attackers behave in multiple stages of an intrusion. As a result, important patterns can be missed, and detection becomes less effective. To address this problem, we built Trappyfi, a multi-service honeypot system designed to observe attacker behaviour in a more realistic environment. The platform runs several decoy services at the same time, allowing it to capture different types

of attack within a single setup.

Trappyfi also combines rule-based detection with a machine learning model to improve accuracy. While rules help catch known attack patterns, the machine learning model helps iden- tify unusual activities that do not match predefined signatures. In addition, the system securely records important events on the blockchain, ensuring that the data remain trustworthy.

By bringing all these components together, Trappyfi pro- vides a clearer and more complete understanding of how modern attacks unfold across multiple systems.

Contributions of this paper:

- A web honeypot that mimics a corporate portal, detecting SQLi, XSS, command injection, path traversal, brute- force, and reconnaissance in real time.
- An SSH honeypot that logs submitted credentials and records a full command log from the attackers simulated shell session.
- A MongoDB honeypot that captures unauthorized con- nections and enumeration attempts.

 DOI : 10.35291/2454-9150.2026.0032

- A fake SMTP server collecting and categorizing phishing and spam messages.
- A cloud credential trap styled as an AWS root login, scoring every submission at maximum risk.
- An Isolation Forest-based ML module for unsupervised, real-time anomaly detection.
- A blockchain audit log using SHA-256 and Ethereum Sepolia for tamper-proof verifiable evidence storage.
- A centralized SOC dashboard with live feeds, session- replay, anomaly views, and dark web credential monitor- ing.
- A real-time Telegram and Email alerting pipeline that transports IP, location, attack type, risk score, and payload for every high-severity event.

II. Literature Survey

This section reviews important research contributions in honeypot-based security systems, with a focus on architec- tural design, deployment approaches, and advanced detection techniques used to study and understand cyber threats.

Cloud-based honeypot deployments have shown that scal- able, programmatic traffic redirection significantly improves the capture of malicious activity [1]. Complementary work has examined how attacker observations feed back into strength- ening network defenses [2], while structured reviews of next-generation systems have recorded architectural differences and their effectiveness against modern threats [3]. The frame of honeypots as proactive rather than reactive instruments — luring adversaries into controlled environments — has gained increasing acceptance [4].

On the technical side, combining static and dynamic anal- ysis improved malicious website detection through client-side honeypots [5]. ThingPot demonstrated that convincing device simulation attracts more sophisticated IoT attacks [6], while adaptive configurations that adjust in response to observed at-tacker behavior yield higher-quality intelligence [7]. Architec- tures focused on maximizing data collection efficiency without proportional cost increases have also been explored [8].

The honeypot-as-a-service delivery model emerged as its own research direction, making distributed enterprise deploy- ments more practical [9]. comparative Cross-cloud evaluations helped practitioners select optimal hosting configurations [10]. Honeyboost showed that the combination of anomaly detection with multi-source data fusion measurably outperforms either approach independently [11]. Low-interaction SSH honeypots, though limited in depth, reliably capture credential campaigns and attacker command patterns [12].

High-interaction systems attract more advanced adversaries and generate richer behavioral data, though at higher opera- tional cost [13]. The cyber deception perspective emphasizes sustained participation for more-in-depth threat actor profil- ing [14]. The Purpose-built HTTP platforms made studying XSS and SQLi campaigns at scale much more tractable [15], and ML integration has advanced automated classification ca- pabilities [16]. More recent work has explored LLM-generated interactions for improved attacker deception [17], [18], and IoT-specific ML-enhanced honeypots show consistent gains in detection adaptability [19], [20].

III. Research Gaps

Looking through this body of work, a few recurring limita- tions stand out quite clearly.

The most fundamental one is that the vast majority of honeypot systems are designed around a single attack surface or a single service type [1]–[4]. This works fine if you only care about one vector, but real attacker campaigns routinely span multiple channels in the same session. A system that only watches SSH will miss the web probing that often precedes it, and a web-only honeypot will not capture the database enumeration that sometimes follows.

Beyond coverage, many existing systems simply do not support real-time monitoring in any meaningful sense. There is no automated alerting, no live dashboard, and no integration with threat intelligence pipelines [5]–[8]. This is a significant operational limitation because the value of honeypot data degrades quickly: knowing that a scan happened yesterday is much less useful than knowing that it is happening right now. Static configurations and hardcoded rule sets are another persistent weakness. Systems built on predefined signatures can only catch attacks that they already know about [9]– [11]. Novel techniques, obfuscated payloads, and slow-moving multi-step campaigns can all slip through undetected. The tension between low-interaction and high-interaction designs has also not been fully resolved in the literature: simpler systems are easy to maintain but provide shallow insight, while more complex interactive systems generate better data
but require significantly more maintenance effort [12]–[14].

Even newer work that incorporates machine learning or large language models tends to present isolated systems with- out connections to centralized monitoring infrastructure or multi-channel alerting [15], [16], [18]. IoT-focused deploy- ments face a related but distinct challenge: they tend to be narrowly scoped by design and do not translate well to

 DOI : 10.35291/2454-9150.2026.0032

general- purpose or heterogeneous environments [19], [20].

Putting these gaps side by side, the case for a platform that combines multi-service deployment, real-time monitoring, automated alerting, anomaly-based detection, and secure audit logging in a single coherent system becomes straightforward. That is the gap Trappyfi is designed to fill.

IV. Proposed System

A. System Architecture Overview

The IHaaS platform is structured as a layered pipeline where every stage has a different responsibility. When an attacker interacts with any part of the system, that activity moves through capture, classification, intelligence enrichment, tamper-proof storage, and finally alerting and visualization — each layer passing to the next in sequence. Figure 1 illustrates how these stages connect. The design was intentional: keeping issues separated makes the platform easier to extend and ensures that a failure in one layer, say the alerting channel, does not compromise the others. The individual components are described below:

- **Fake Corporate Entry Environment:** The outermost layer of Trappyfi is a convincing replica of a Thomas Cook Travel portal. The choice of a recognizable cor- porate brand was deliberate — an attacker who be- lieves they are checking a real company target tends to behave more naturally and use their actual toolset, instead of holding back due to suspicion. The portal provides endpoints that attackers naturally test: `/login`, `/admin`, `/api`, `/payments`, `/backup.zip`, and `/config.php`. These are not randomly chosen — they replicate the paths that automated scanners and experi- enced manual attackers look for on real web applications, which is exactly the kind of organic engagement the system is designed to attract.
- **Multi-Service Honeypot Layer:** Five false services run in parallel, each targeting a different attack vector that a real campaign might use. The *Web Honeypot* is an HTTP service that closely inspects every incoming request — bodies, URL parameters, and headers — for signatures of XSS, SQL injection, path traversal, command injection, and automated reconnaissance. Nothing passes through without being recorded with complete details.

  The *SSH Honeypot*, built on the Paramiko library, listens on port 2222 and presents itself as a functional interactive shell. Every credential pair an attacker submits during authentication is recorded, and once they believe they are inside, a complete transcript of every command they type is preserved for later analysis.

  The *Email Honeypot* uses the aiosmtpd framework to run a fake SMTP server on port 2525. It accepts incoming messages without filtering, storing sender details, subject lines, and full message bodies, which makes it a reliable collector of phishing and spam campaigns targeting the fictional company.

  The *Database Honeypot* presents itself as an accidentally exposed MongoDB instance on port 27017 — a miscon- figuration that attackers actively scan for because it is so common in real deployments. Every connection attempt and query check is logged in detail. The *Cloud Credential Trap*, accessible at `/aws-console`, mimics the AWS management console login page. Any root credentials submitted there are captured immediately and assigned the maximum risk score of 10, since there is no doubt about the intent behind that action.
- **Deception File Layer:** Spread across the environment are several honeyfiles: `backup.zip`, `api_keys.env`, `employees.csv`, and `database.sql`. These files are not functional - they exist purely as trap, named and placed to attract the kind of attacker who is specifically looking for data to steal. Any access to them is treated as a confirmed high-risk signal. The false-positive rate from this mechanism is effectively zero, because no legitimate user or process in the environment has any reason to touch these files.
- **Attack Detection Engine:** Every captured event passes through a rule-based classifier that checks request pay- loads, parameters, and headers against a selected set of known attack signatures, which are listed in Table I. After classification, each event is assigned a risk score on a scale of one to ten. Scores from one to three are labeled LOW, four to six are MEDIUM, and seven to ten are HIGH. The score is not decided by attack category alone — it also considers how frequently the same source IP has appeared, and how sensitive the targeted endpoint is within the simulated environment.

 DOI : 10.35291/2454-9150.2026.0032

TABLE I
ATTACK DETECTION SIGNATURES

Attack Category	Detection Pattern
XSS	¡script¿, ¡svg onload=
SQL Injection	OR 1=1, UNION SELECT
Command Injection	&&, ; id, ; whoami
Path Traversal	../, /etc/passwd
Credential Theft	AWS ROOT, api key=
Reconnaissance	sqlmap, Nikto, dirsearch

- **Machine Learning Anomaly Detection:** Rule-based matching has a built-in ceiling: it only catches what someone already thought to write a signature for. To push past that ceiling, we deployed an Isolation Forest model that runs in parallel with the rule engine and evaluates multiple behavioral features of each incoming event. We chose Isolation Forest specifically because it is unsupervised — it does not need a labeled dataset of attacks to get started. Instead, it builds a model of what normal traffic looks like and marks anything that is very different from normal behaviour. The model is set to retrain automatically as new data comes in, so it stays fairly accurate as traffic patterns change over time instead of becoming outdated.
- **Threat Intelligence Pipeline:** Before any event is saved to storage, it goes a step that adds more details that adds geographic context from the source IP, the associated ASN and ISP, a parsed user-agent string, and device classification information. The detailed record is stored as a structured JSON document in a MongoDB Atlas collection. (`honeypot_db.attacks`), where it is im- mediately available for dashboard queries and for feeding downstream analysis pipelines.
- **Blockchain Audit Log:** The integrity of collected ev- idence matters, especially in contexts where the data might feed into an incident report or a legal proceeding. To address this, each stored event is hashed with SHA- 256 and the resulting hash is submitted to the Ethereum Sepolia test network. This creates a record that can be checked easily: if someone changes a stored event later, the new hash will not match the one saved on the blockchain, and the change will be noticed right away.
- **Alerting and Reporting:** When an event gets a HIGH score, the system does not wait for a human to notice it on the dashboard. An alert fires immediately to both a desig- nated Telegram channel and an email inbox, carrying the source IP, geographic location, attack type, numerical risk score, and a sample of the captured payload. For the most critical events, a PDF incident report is also generated automatically, ready to be attached to a response ticket without any manual preparation.
- **SOC Monitoring Dashboard:** A Flask-based application ties all of these components together into a single inter- face. It delivers a live attack feed, an interactive world map that plots attack origins in real time, timeline charts, statistical summaries, and a session replay feature that lets an analyst go through an attacker's recorded behavior chronologically. A separate dark web monitoring panel completes the dashboard, checking for any leaked cre- dentials related to the honeypot environment on hidden forums or paste sites.

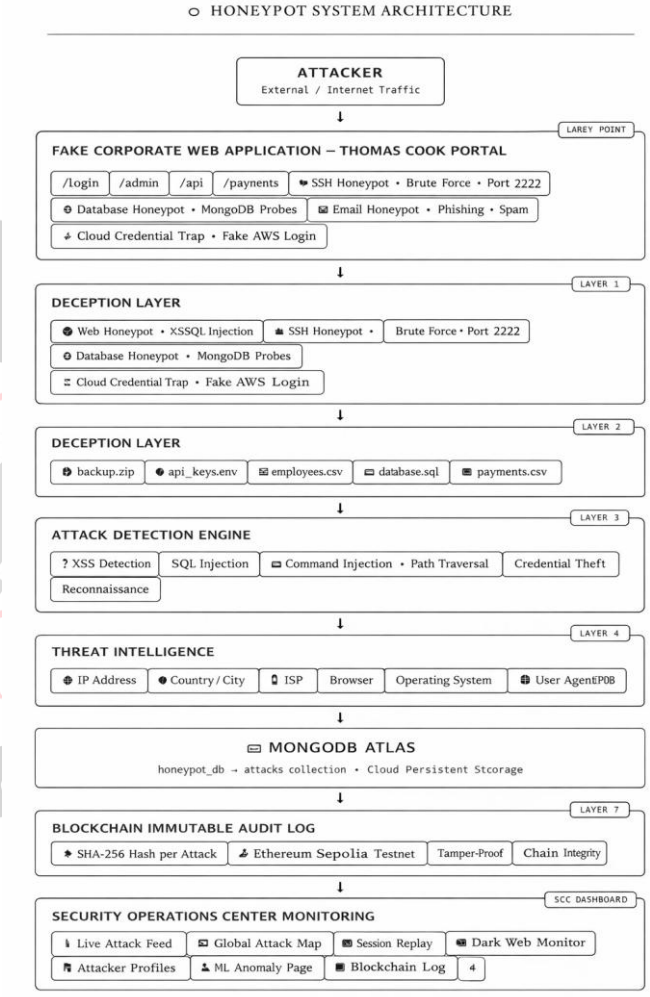

Fig. 1. IHaaS System Architecture

B. System Workflow

IHaaS runs as a continuous loop. Traffic hits whichever hon- eypot surface the attacker tests; the detection engine classifies and scores it; the enrichment pipeline adds geographic and contextual metadata; the event is stored in MongoDB Atlas and simultaneously hashed to the blockchain. HIGH score events trigger alerts within seconds and the dashboard updates in near real time. Figure 2 trace this flow from end to end.

 DOI : 10.35291/2454-9150.2026.0032

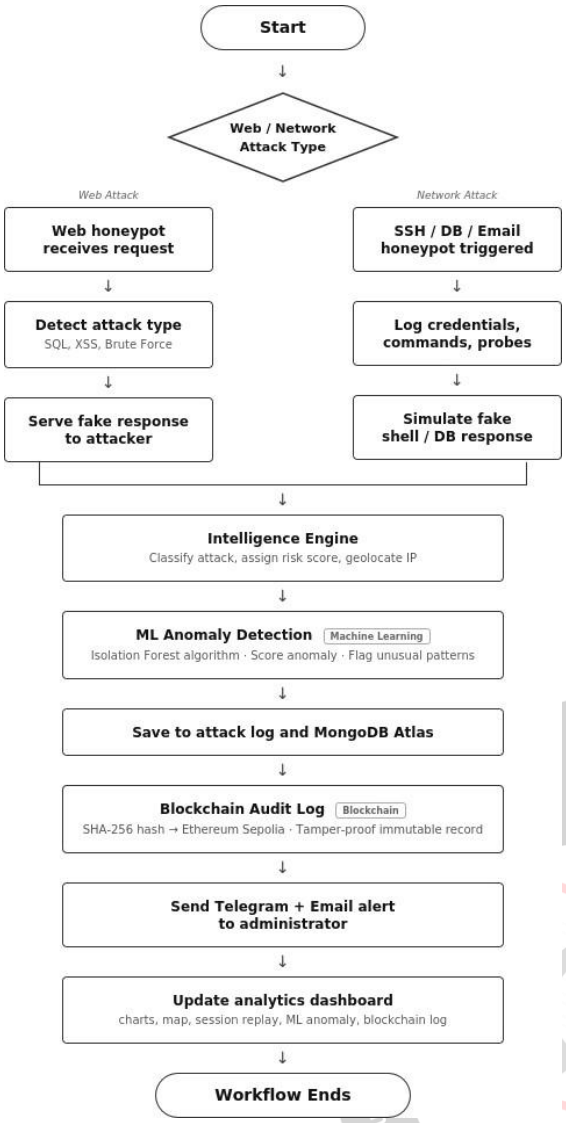

Fig. 2. IHaaS Workflow Diagram

V. Experimental Results and Analysis

A. Results

We deployed Trappyfi on a cloud instance and left it exposed to real internet traffic for 30 consecutive days without intervention or filtering. During the deployment period, the system recorded 1,001 distinct attack events from sources spread across 20 countries. The Isolation Forest anomaly detection module detected 312 of these events as unusual, which is equivalent to a detection rate of 31.2%. Every cloud credential theft attempt that occurred during the test period was among those detected, as was the majority of activity that occurred outside normal business hours- a behavioral signal that the rule engine does not score particularly heavily on its own. All 1,001 events were preserved in the blockchain- backed log. Ethereum Sepolia was successfully connected for 87 events during testing, and the blockchain remained fully secure with no integrity issues at any time.

TABLE II
Attack Event Distribution by Honeypot Service

Honeypot Service	Events	High Risk	Top Attack
Web Honeypot	871	407	XSS / SQLi
SSH Honeypot	57	42	Brute Force
DB Honeypot	15	15	DB Enumeration
Email Honeypot	48	21	Phishing
Cloud Credential Trap	10	10	Credential Theft
Total	**1001**	**495**	–

Web based attacks made up 87% of all traffic, which is consistent with what broader industry threat reports have been saying for several years - web applications are the most accessible target area for the widest range of attackers. Events classified as HIGH risk, those with a score of seven or more, made up 495 out of 1,001 cases, or 49.5% of the total. Every attempt to steal the cloud credential scored the maximum score of 10 out of 10. The mean risk score in all events was 6.7. The ML model produced zero false positives in separately verified normal traffic. The alert delay was approximately about 1.4 seconds on average, and the dashboard was updated in approximately 800 milliseconds with 50 users at the same time.

B. Comparative Feature Analysis

To understand the features of Trappyfi more clearly, Ta- ble III shows a feature-based comparison of Trappyfi with Cowrie and T-Pot, highlighting its unique capabilities.

TABLE III Comparative Feature Analysis

Feature	Cowrie	T-Pot	Trappyfi
Multi-Service Support	No	Yes	Yes
Automatic Classification	No	Partial	Yes
Threat Intelligence	No	Partial	Yes
Real-Time Dashboard	No	Yes	Yes
Honeyfile Detection	No	No	Yes
Cloud Credential Trap	No	No	Yes
ML Anomaly Detection	No	No	Yes
Blockchain Audit Log	No	No	Yes
Session Replay	No	No	Yes
Dark Web Monitoring	No	No	Yes

Cowrie is intentionally designed to emulate only SSH and Telnet, so it does not support multiple services by design. T-Pot uses a broader approach by combining multiple honeypot tools into a single deployment, but several of its most practically useful features such as- automatic classification, threat intelli- gence enrichment- remain only partially implemented. Neither system offers honeyfile-based detection, cloud credential trap, ML-driven anomaly detection, tamper-proof audit log, session replay, or dark web monitoring. Trappyfi provides all ten features in a unified platform.

C. Discussion

All five services received real traffic during the 30-day run, which shows that attackers do not stick to just one method. A web-only deployment would have covered 87% of events but left SSH brute-force

 DOI : 10.35291/2454-9150.2026.0032

activity, database enumeration, phishing captures, and every cloud credential theft attempt completely invisible. Each of those gaps represents a category of attacker behavior that tells a different part of the story.

Honeyfile access basically produced no false positives be- cause nothing real ever touches those files. Any access to the honeyfile is considered malicious, making it one of the most reliable high-confidence signals the system generates — exactly the kind of indicator that matters most when analysts are overwhelmed with lower-quality alerts.

What was really interesting was the interplay between the rule engine and the ML module. Several of the 312 detected unusual events had scored only medium risk using the signature-based classifier because the individual data looked normal on its own. But the Isolation Forest, considering timing, past behavior from the same IP, and the sequence of endpoints accessed, correctly identified them as unusual. That layer of contextual reasoning is something that pure signature matching simply cannot replicate.

Geolocation and ISP data turned raw log entries into some- thing an analyst can actually reason about quickly. Blockchain verification was held cleanly across all stored records, with no integrity violations detected. The alert delay stayed around 1.4 seconds, and the dashboard was updated in less than 800 milliseconds at peak load — fast enough to be truly useful in real-world use.

VI. Limitations and Future Work

The signature-based engine cannot detect hidden data or techniques that are not part of its current rules — this is a natural limitation of any pattern-matching system. The Isolation Forest model works well when traffic is stable, but its fixed settings can become inaccurate over time and need regular adjustment. The simulated environment, while realistic, cannot fully match the complexity of a real enterprise network, so some attacker behaviors may not appear the same as they would on a real system. Ethereum Sepolia recording is currently used only for selected events; at a large scale, grouping event hashes using a Merkle tree would be a practical way to reduce blockchain transaction costs.

Another limitation worth naming is the relatively short evaluation window. Thirty days gave us enough traffic to test the core system, but it is not long enough to say much about how the platform performs over months of continuous operation. Attack patterns change over time, new tools appear, and attackers keep improving their methods — all of which can affect how well the detection system performs. A longer longitudinal study would give a much clearer picture of accuracy drift, alert fatigue, and whether the ML module retraining schedule is practical as the data volume grows. We consider the current deployment as a proof of concept, not a fully ready production system.

There is also the question of the detection depth in the SSH and database honeypots. Right now, both services log what they see but do not attempt to engage attackers beyond the initial interaction. A more advanced SSH honeypot that simulates a richer shell environment with fake files, realistic directories, and delayed error responses would likely keep attackers engaged longer and reveal more about their behavior after gaining access. Similarly, the MongoDB honeypot could respond to queries with synthetic data rather than simply logging and dropping the connection, which might reveal more about what an attacker is actually trying to extract and how they plan to use it.

Moving forward, the most useful additions would be LSTM- based temporal detection that reasons across full session sequences rather than individual events, automated response capabilities such as dynamic IP blocking and adaptive decep- tion, decentralized audit logging via smart contracts, external threat intelligence feed integration, and SIEM connectivity for broader enterprise deployment.

VII. Conclusion

Trappyfi was built around a straightforward observation: attackers move across multiple surfaces, so a honeypot worth deploying should do the same. The platform runs five de- ceptive services in parallel, processes everything through a shared detection and enrichment pipeline, and surfaces the result through a live SOC dashboard with sub-second alerting. Two things differentiate it from existing tools: an unsupervised Isolation Forest module that catches behavioral anomaly sig- natures that would be missed, and a blockchain audit log that makes every captured record independently verifiable.

During 30 days of real internet exposure, the system recorded 1,001 attacks from 20 countries, the anomaly detector reached a detection rate of 31.2% with zero false positives in clean traffic, and the entire evidence chain remained intact. It caught every cloud credential theft attempt and identified off-hours activity that rule-based scoring alone would have underrated. For teams looking for a honeypot platform that goes beyond passive collection and actually supports investi- gation and evidence preservation, Trappyfi offers a practical and extensible starting point.

In general, the proposed framework shows strong potential as as scalable and efficient solution for advanced threat detec- tion and cyber intelligence generation. Future improvements focused on adaptive

 DOI : 10.35291/2454-9150.2026.0032

learning, decentralized logging, and large- scale deployment are expected to further enhance its performance in rapidly evolving threat environments.

REFERENCES

[1] Z. Moric, V. Dakic, and D. Regvart, "Advancing cybersecurity with honeypots and deception strategies," *Informatics*, vol. 12, no. 1, p. 14, 2025.

[2] S. Lanz, S. L.-R. Pignol, P. Schmitt, H. Wang, M. Papaioannou,
G. Choudhary, and N. Dragoni, "Optimizing internet of things honeypots with machine learning: a review," *Applied Sciences*, vol. 15, no. 10, p. 5251, 2025.

[3] H. T. Otal and M. A. Canbaz, "Llm honeypot: leveraging large language models as advanced interactive honeypot systems," 2024.

[4] X. Yang, J. Yuan, H. Yang, Y. Kong, H. Zhang, and J. Zhao, "A highly interactive honeypot-based approach to network threat management," *Future Internet*, vol. 15, no. 4, p. 127, 2023.

[5] M. Rabzelj, L. S. Juznic, M. Volk, A. Kos, M. Kren, and U. Sedlar, "De- signing and evaluating a flexible and scalable http honeypot platform," *Electronics*, vol. 12, no. 16, p. 3480, 2023.

[6] D. A. Firmansyah and A. Zahra, "Honeypot-based threat detection using machine learning techniques," *International Journal of Engineering Trends and Technology*, vol. 71, no. 8, pp. 243–252, 2023.

[7] D. Zielinski and H. A. Kholidy, "An analysis of honeypots and their impact as a cyber deception tactic," 2023.

[8] S. C. V, A. Anand, and Muskan, "Honeypots: an incentive approach to modern security," 2023.

[9] S. Pahal and P. Priya, "A comprehensive research study on low- interaction secure shell honeypot," *Mapana Journal of Sciences*, vol. 21, no. 4, pp. 99–118, 2022.

[10] S. Kandanaarachchi, H. Ochiai, and A. Rao, "Honeyboost: boosting honeypot performance with data fusion and anomaly detection," 2021.

[11] C. Kelly, N. Pitropakis, A. Mylonas, S. McKeown, and W. J. Buchanan, "A comparative analysis of honeypots on different cloud platforms," *Sensors*, vol. 21, no. 7, p. 2433, 2021.

[12] J. H. Jafarian and A. Niakanlahiji, "Delivering honeypots as a service," 2020.

[13] W. Fan, Z. Du, M. Smith-Creasey, and D. Fernandez, "Honeydoc: an efficient honeypot architecture enabling all-round design," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, 2019.

[14] D. Fraunholz, M. Zimmermann, and H. D. Schotten, "An adaptive honeypot configuration, deployment and maintenance strategy," 2019.

[15] M. Wang, J. Santillan, and F. Kuipers, "Thingpot: an interactive internet- of-things honeypot," 2018.

[16] T.-M. Koo, H.-C. Chang, Y.-T. Hsu, and H.-Y. Lin, "Malicious website detection based on honeypot systems," 2013.

[17] V. Sharma, "Use of honeypots to increase awareness regarding network security," *International Journal of Recent Technology and Engineering*, 2012.

[18] R. M. Kinariwala and G. B. Jethava, "Research on various next gener- ation honeypot systems," 2012.

[19] Y. K. Jain and S. Singh, "Honeypot based secure network system," 2011.

[20] M. Balamurugan and B. S. C. Poornima, "Honeypot as a service in cloud," *International Journal of Computer Applications*, 2011.

 DOI : 10.35291/2454-9150.2026.0032