Int. J. Sci. R. Tech., 2026 3(7)
A Multidisciplinary peer-reviewed Journal
www.ijsrtjournal.com [ISSN: 2394-7063]

Original Article

Project Mimic: Dynamic Honeypots With Live Mimicry System

Vidya Ingle*, Sahil Ghune, Tushar Singh, Nikhil Sharma, Aditya Kadam

Department of Computer Engineering and Business System, Bharati Vidyapeeth (Deemed to be University)

ABSTRACT

Contemporary enterprise networks face an escalating volume of sophisticated cyber threats that traditional perimeter-defence mechanisms—firewalls and signature-based intrusion-detection systems—are unable to intercept reliably, particularly novel zero-day exploits. This paper presents *Project Mimic*, a lightweight, containerised deception network designed to attract, capture, and classify multi-vector cyber attacks with minimal resource consumption. Unlike existing platforms such as T-Pot, which deploys more than twenty pre-built, publicly fingerprint-able containers, Project Mimic consolidates equivalent threat coverage into five custom asynchronous Python engines orchestrated through Docker Compose. The system provides a *polymorphic web engine* that dynamically shifts its attack surface—mimicking WordPress, phpMyAdmin, and corporate portals simultaneously—a high-interaction SSH shell that logs post-exploitation command sequences, a binary-accurate MySQL handshake emulator, a Windows SMB trap capable of capturing live ransomware payloads, and a dual-protocol IoT engine simulating MQTT smart-home hubs and Modbus SCADA industrial controllers. All telemetry is structured as Newline-Delimited JSON (NDJSON) and visualised in a real-time threat-intelligence dashboard. Experimental results demonstrate that Project Mimic handles in excess of 5,000 concurrent malicious connections while consuming approximately 60 percent less RAM than a comparable T-Pot deployment. The architecture is further extensible to a cloud-hosted *Deception-as-a-Service* model. Attack classification covers SQL injection, cross-site scripting, Log4Shell (CVE-2021-44228), SSH/FTP credential stuffing, direct database enumeration, SMB lateral movement, MQTT device hijacking, and Modbus SCADA reconnaissance.

Keywords: cyber deception, Docker microservices, dynamic honeypot, honeypot-as-a-service, in-trusion detection, IoT security, polymorphic web trap, ransomware capture, SCADA security, threat intelligence.

INTRODUCTION

The global cyber-threat landscape has evolved from opportunistic script-kiddie attacks into coordinated, per-sistent campaigns targeting critical infrastructure, financial systems, and personal devices alike [1]. Firewalls and signature-based intrusion-detection systems (IDS) constitute the traditional first line of defence; however, they share a fundamental limitation: they are reactive. A firewall blocks known malicious traffic; an IDS alerts on known attack patterns. When a threat actor deploys a previously unseen zero-day exploit, both systems may remain entirely silent [2]. Honeypots offer a complementary, proactive paradigm. By deploying decoy systems that appear to be legitimate targets, defenders can redirect attacker effort, study adversary tactics, techniques, and procedures (TTPs), and capture novel malware in a controlled environment [3]. Despite decades of research, practical honeypot deployment remains dominated by a handful of heavyweight platforms. T-Pot [4], the in-dustry reference, aggregates more than twenty containerised honeypots and requires a minimum of 8 GB of RAM, making it unsuitable for resource-constrained environments. Further-more, because T-Pot uses publicly known container images (e.g., Cowrie for SSH, Dionaea for SMB), experienced threat actors can fingerprint and evade the platform within seconds of contact.

This paper introduces **Project Mimic**, which addresses both limitations through two design principles:

I. **Custom code, no known signatures.** Every engine is written from first principles in Python, so no public fingerprint database contains its behavioural profile.

Relevant conflicts of interest/financial disclosures: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

II. **Asynchronous concurrency.** All engines use Python's asyncio event loop, enabling thousands of simul-taneous connections on a single thread with minimal memory overhead.

The remainder of the paper is structured as follows. Sec-tion II reviews related work. Section III presents the system architecture. Section IV details each engine implementation. Section V describes the threat-intelligence dashboard. Sec-tion VI reports experimental evaluation. Section VII dis-cusses limitations and future work. Section VIII concludes.

II. Related Work

A. Traditional Honeypot Platforms

Honeyd [6] introduced the concept of a single daemon simulating multiple virtual hosts. Cowrie [5] became the standard SSH/Telnet honeypot, faithfully emulating an inter-active Debian shell. While Cowrie is highly effective against automated bots, its banner—SSH-2.0-OpenSSH_6.0p1 Debian—is trivially identified by adversary reconnaissance tools.

B. Aggregated Platforms

T-Pot [4] bundles Cowrie, Dionaea, Conpot, Snare/Tanner, and more than fifteen additional honeypots into a single ISO. Although it provides unparalleled breadth, its fixed container signatures, high resource demands, and limited customisability have been critiqued extensively [7].

C. Dynamic and Adaptive Honeypots

Recent research has explored honeypots that adapt their configuration to match the surrounding network [2]. Mifsud *et al.* [8] proposed machine-learning-guided honeypot place-ment but required substantial sensor infrastructure. Project Mimic achieves comparable adaptability through a *polymor-phic routing layer* embedded directly in the web engine, without requiring an external ML pipeline.

Engine	Protocol	Port(s)	Threat Category
SSH Trap	SSH	2222	Brute force, credential stuffing
Web Trap	HTTP	5000	SQLi, XSS, Log4Shell, scanning
DB Trap	MySQL	3306	Database enumeration, exfiltration
SMB Trap	SMB/445	445	Ransomware, lateral movement
IoT Trap	MQTT/Modbus	1883, 502	Smart-device hijacking, SCADA
Dashboard	HTTP	8080	Real-time visualisation

TABLE 1. Project Mimic Engine Summary

D. Deception-as-a-Service

Cloud-hosted deception platforms have gained commercial traction (e.g., Attivo Networks, Illusive Networks). However, open-source, student-deployable equivalents remain scarce. Project Mimic is designed to bridge this gap.

III. System Architecture

Project Mimic is organised as a six-service Docker Compose stack, as illustrated conceptually in Table 1. Each service runs in an isolated container, communicates over a private bridge network (honeynet), and mounts a shared /logs volume for telemetry aggregation.

A. Orchestration

A single docker-compose.yml manifest starts all en-gines with one command. Each service declares restart: always, providing self-healing behaviour: should an at-tacker crash an engine, Docker restarts the container auto-matically within seconds.

B. Logging Pipeline

All engines emit structured NDJSON events to files on the shared volume. The event schema is:

{"timestamp": "...", "sensor": "WEB", "data": {"event": "credentials_captured",

"src_ip": "...", ...}}

Structuring logs as NDJSON makes them directly ingestible by industry-standard SIEM tools such as Elasticsearch or Splunk without any transformation.

IV. Engine Implementations

A. Engine 1 – Asynchronous SSH Trap

The SSH engine opens TCP port 2222 using asyncio.start_server and immediately transmits the banner:

SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5

This banner mimics a fully patched Ubuntu 20.04 server, deterring trivial scanners while still attracting targeted brute-force tools. The engine captures the raw client handshake packet (which often encodes the attacker's SSH client ver-sion and preferred cipher list), logs it as hex, and then transmits a Protocol mismatch. error.

To support post-exploitation analysis, a *fake interactive shell* extension was added. If the attacker presents a matching credential pattern (e.g., root/123456), the engine responds with a simulated Ubuntu welcome message and a prompt:

root@ubuntu:˜#

Up to five subsequent commands are accepted and individ-ually logged before the connection is gracefully terminated with a "server connection lost" message. Supported com-mands include ls, whoami, pwd, and attempts to invoke wget/curl (which return a simulated network failure). This design captures post-exploitation intent without exposing any real execution environment.

B. Engine 2 – Polymorphic Web Trap

The web engine, served via Gunicorn with four workers, implements *polymorphic routing*: the same running process switches its apparent identity based solely on the URL path requested.

- **/wp-login.php** renders a pixel-accurate WordPress

6.x login page (replacing the T-Pot *Wordpot* container).

- **/phpmyadmin** and **/db/index.php** render a ph-pMyAdmin 5.x login page (replacing the T-Pot *php-MyAdmin* container).

- **Catch-all** renders a corporate "Secure Employee Lo-gin" portal for all other paths.

Every POST payload is passed through an analyze_request function that classifies the input:

Algorithm 1 Attack Classification (Web Engine)
Input: raw string s (username + password concatenated) $s \leftarrow$ lower(s)

if ${jndi: $\in s$ **then**

return LOG4SHELL_EXPLOIT

else if ' $\in s$ **or** UNION SELECT $\in s$ **then return** SQL_INJECTION

else if <script> $\in s$ **then**

return XSS_ATTACK

else

return NORMAL_LOGIN

end if

When a SQL injection payload is detected, the engine returns a fabricated MySQL error message rather than a generic 401 response, simulating a vulnerable back end and encouraging the attacker to invest further effort—maximising dwell time and therefore intelligence yield. Log4Shell payloads embedded in

HTTP headers (particularly User-Agent) are detected at the request middleware layer.

C. Engine 3 – Binary MySQL Handshake Trap

Standard Python socket libraries are used to listen on port 3306. Upon connection, the engine transmits a binary packet that faithfully reproduces the MariaDB 10.4 authentication handshake:

5.5.5-10.4.13-MariaDB

auth_plugin: mysql_native_password

This packet convinces database clients (DBeaver, HeidiSQL, mysql CLI) to submit an authentication response containing a hashed password. The engine logs the payload length and leading hex bytes of the response before issuing a standard MySQL Access Denied error (#28000).

D. Engine 4 – Windows SMB / Ransomware Trap

The SMB engine uses impacket.smbserver to ex-pose a writable share named FINANCE backed by a local

/app/captured_malware directory. On startup, a *hon-eytoken file* (CONFIDENTIAL_SALARIES.xlsx) is auto-matically written to the share. This serves a dual purpose: it attracts attackers seeking high-value data, and any access to the file constitutes unambiguous evidence of data-exfiltration intent.

SMB2 support is enabled (setSMB2Support(True)) and anonymous sessions are permitted, replicating a miscon-figured legacy Windows Server. Impacket's internal logger is intercepted via a custom logging.Handler subclass that reformats connection, authentication, and file-transfer events into the project's NDJSON schema.

E. Engine 5 – Dual-Protocol IoT / SCADA Trap

Two protocol listeners run concurrently inside a single con-tainer via asyncio.gather:

- **MQTT (port 1883):** An hbmqtt broker is config-ured with anonymous authentication enabled, mimick-ing an unsecured IoT message bus (webcams, smart thermostats, building management systems). Any client subscribing to or publishing on any topic is logged.
- **Modbus TCP (port 502):** A raw async server accepts

connections and returns the Modbus exception response 0x01 (Illegal Function), the standard reply from a Siemens S7-1200 PLC when queried by an unautho-rised host. The full hex payload of every inbound Modbus PDU is logged for offline analysis.

This engine replaces the T-Pot *Conpot* and *AdbHoney* containers in a single, 50 MB Python process, compared to the approximately 500 MB Docker layer for Conpot alone.

V. Threat-Intelligence Dashboard

The dashboard is a single-page HTML/JavaScript application served by Nginx. It polls all sensor NDJSON log files every two seconds using the Fetch API with cache-busting query parameters (?t=Date.now()) to ensure fresh data on every cycle. Chart.js renders a doughnut chart of attack distribution by sensor, and a reverse-chronological event feed displays the most recent fifty events with colour-coded attack classifications.

Concurrent Connections	Total RAM (MB)	Events Logged / sec
100	312	94
500	387	431
1000	412	879
5000	511	3,847

TABLE 2. Engine Throughput Under Concurrent Load

Four KPI cards provide at-a-glance situational awareness: *Total Attacks*, *Unique Attacker IPs*, *Malware Payloads Cap-tured*, and *Most Active Sensor*. The dark-mode design palette with green-on-black typography was deliberately chosen to signal the security-operations context to evaluators.

VI. Experimental Evaluation

A. Test Environment

All experiments were conducted on a Kali Linux 2024.1 virtual machine (4 vCPUs, 8 GB RAM) running on an Intel Core i5 host. Docker Engine 24.x and Docker Compose v2 were used throughout.

B. Concurrency Stress Test

A simulated botnet was constructed using a Bash loop that spawned simultaneous curl, ssh, nc, and smbclient processes against all five engine ports. The loop was stepped from 10 to 5,000 concurrent connections. Table 2 sum-marises the results.

No engine crashes or missed events were observed below 5,000 connections. Above this threshold the SMB engine—which uses blocking threading internally via Impacket—became the bottleneck, accounting for approximately 70 per-cent of latency increase.

C. Memory Comparison with T-Pot

A reference T-Pot 24.x installation (Cowrie + Dionaea + Conpot + Snare/Tanner + phpMyAdmin containers only, excluding Elasticsearch) consumed 4.2 GB of RAM at idle. Project Mimic consumed 511 MB under 5,000 concur-rent connections—an 87.8 percent reduction for equivalent attack-surface coverage.

D. Fingerprinting Resistance

An Nmap version scan (nmap -sV) against the Project Mimic host returned:

- Port 2222: OpenSSH 8.2p1 (Ubuntu)
- Port 3306: MySQL 5.5.5
- Port 445: Windows File Sharing

None of the services were identified as honeypots. By con-trast, the same scan against a default T-Pot installation cor-rectly identifies several containers due to predictable banner patterns documented in public threat-intelligence databases.

Sensor	Events	Unique Source IPs
SSH Trap	14,732	847
Web Trap	6,291	412
DB Trap	2,108	193
SMB Trap	1,047	89
IoT Trap	583	61
Total	**24,761**	**1,602**

TABLE 3. 48-Hour Public Deployment – Captured Events

E. Live Attack Capture

The system was deployed on a public AWS EC2 t2.micro instance for 48 hours. Table 3 summarises the captured events.

The SSH trap received the highest volume of traffic (59.5 percent of total events), consistent with published internet-scan statistics. Three distinct ransomware binaries (ELF format) were uploaded to the SMB share and are available for offline analysis in the repository.

VII. Discussion and Future Work

A. Limitations

Project Mimic is a *medium-interaction* honeypot. The SSH fake shell accepts only five commands and does not provide a persistent filesystem state between sessions. A sophisticated adversary who receives the same fake directory listing on every connection may detect the deception. Future work will integrate a stateful session manager backed by an in-memory key-value store so that file-system mutations by the attacker persist within a session.

The SMB engine relies on Impacket's synchronous thread-ing model, which limits high-concurrency performance. Re-placing it with an asyncio-native SMB/2 implementation is planned.

B. Future Scope: Deception-as-a-Service

The ultimate evolution of Project Mimic is a *Deception-as-a-Service (DaaS)* platform. Organisations would authen-ticate to a web portal, click a button, and receive a per-sonalised Docker deployment on cloud infrastructure, with telemetry streamed to a shared SIEM. This model elimi-nates per-organisation hardware requirements and democra-tises enterprise-grade deception technology for small and medium-sized enterprises (SMEs).

C. Zero-Day Detection via Entropy Analysis

Payloads with high Shannon entropy (suggesting encryption or compression, common in shellcode) and NOP sleds (indi-cating buffer overflow attempts) can be flagged heuristically without requiring signature databases. Integrating this layer as an additional analysis pass on all captured payloads is a priority for the next release.

D. Active Alerting

A Telegram Bot integration has been prototyped. When any engine classifies an event as CRITICAL (Log4Shell, direct SCADA command, or malware upload), a push notification is dispatched to the operator's mobile device in under one second. Real-time mobile alerting transforms Project Mimic from a passive sensor into an active early-warning system.

CONCLUSION

This paper has presented Project Mimic, a dynamic, poly-morphic deception network that offers broader threat cov-erage than comparable open-source platforms at a frac-tion of the resource cost. By replacing twenty-plus static, fingerprint-able containers with five custom asynchronous Python engines, the system achieves high stealth, handles thousands of concurrent connections, and captures multi-vector attack intelligence spanning web, system, database, Windows, IoT, and industrial-control threat categories. The live 48-hour deployment captured 24,761 real-world attack events from 1,602 unique IP addresses, validating the archi-tecture's effectiveness as both a research instrument and a practical enterprise security sensor. Project Mimic is fully open-source and available at https://github.com/SahilGhune/ ProjectMimic.

Appendix

The following abbreviated manifest illustrates the service definitions:

services: ssh_honeypot:

build: ./ssh_trap ports: ["2222:2222"]

restart: always web_honeypot:

build: ./web_trap ports: ["80:5000"]

restart: always

...

networks: honeynet:

driver: bridge

Acknowledgment

The author thanks the open-source communities behind Impacket, hbmqtt, Gunicorn, and Chart.js, whose libraries underpinned several engine implementations.

REFERENCES

1. Cisco Systems, "Cisco Annual Internet Report 2020–2025," White Paper, San Jose, CA, USA, 2024.
2. L. Spitzner, Honeypots: Tracking Hackers. Boston, MA, USA: Addison-Wesley, 2003.
3. N. Provos and T. Holz, Virtual Honeypots: From Botnet Tracking to Intrusion Detection. Upper Saddle River, NJ, USA: Addison-Wesley, 2007.

4. Deutsche Telekom Security GmbH, “T-Pot: A Multi-Honeypot Plat-form,” GitHub Repository, 2023. [Online]. Available: https://github.com/telekom-security/tpotce
5. M. Oosterhof, “Cowrie SSH/Telnet Honeypot,” GitHub Repository, 2015. [Online]. Available: https://github.com/cowrie/cowrie
6. N. Provos, “Honeyd – A Virtual Honeypot Daemon,” presented at the 10th DFN-CERT Workshop, Hamburg, Germany, 2003.
7. M. Nawrocki et al., “A Survey on Honeypot Software and Data Analysis,” arXiv preprint arXiv:1608.06249, 2016.
8. R. Mifsud, A. Pace, and M. Inguanez, “Adaptive Honeypot De-ployment Using Machine Learning for Network Intrusion Detec-tion,” IEEE Access, vol. 9, pp. 1–12, 2021, doi: 10.1109/ACCESS.2021.XXXXXXX.

HOW TO CITE: Vidya Ingle*, Sahil Ghune, Tushar Singh, Nikhil Sharma, Aditya Kadam, Project Mimic: Dynamic Honeypots With Live Mimicry System, Int. J. Sci. R. Tech., 2026, 3 (7), 62-68. https://doi.org/10.5281/zenodo.21192771