International Journal of Creative and Open Research in Engineering and Management
ISSN: 3108-1754 (Online)
Volume 02 Issue 11 November-2026 | Impact Factor: 3.5



Honeypot-Based Network Security Monitoring: Design, Implementation and Simulation-Driven Evaluation of a T-Pot Deployment with a Machine-Learning-Ready Control Center

Bhimashankar USN: 1AK23CS021
Beerappa USN: 1AK24CS017
Ambareesh Gadikar USN: 1AK24CS007
Abhishek Helawar USN: 1AK24CS004
Akshaya Institute of Technology, Tumakuru, Karnataka, India
Department of Computer Science and Engineering
Guide: **Prof. Thippamma S**, Associate Professor, Department of CSE
Visvesvaraya Technological University, Belagavi, Karnataka, India

How to Cite this Article:
Bhimashankar, , Beerappa, , Gadikar, A. & Helawar, A. (2026). Honeypot-Based Network Security Monitoring: Design, Implementation and Simulation-Driven Evaluation of a T-Pot Deployment with a Machine-Learning-Ready Control Center. International Journal of Creative and Open Research in Engineering and Management, 02(11), 1-9. https://doi.org/10.55041/ijcope.v2i11.157
License:
This article is published under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.
© The Author(s). Published by International Journal of Creative and Open Research in Engineering and Management.





https://doi.org/10.55041/ijcope.v2i11.157

***Abstract*—Signature-driven intrusion detection systems raise many false alarms and cannot recognise attacks for which no signature exists. Honeypots offer a complementary approach: because a decoy has no legitimate users, every interaction with it is suspicious, so the resulting telemetry is both low-noise and rich in attacker behaviour. This paper presents the design and implementation of a honeypot-based network security monitoring system built on the open-source T-Pot platform, which combines the Cowrie, Dionaea, Honeytrap and Conpot decoys with the Suricata intrusion detection engine and an Elasticsearch–Logstash–Kibana analytics stack. The deployment is hosted in an isolated virtual machine with strict outbound containment. On top of the platform we developed Python traffic simulators that generate brute-force, reconnaissance and benign sessions against a local honeypot, together with a web-based Control Center for launching simulations and exploring captured sessions, and a dataset-export path intended for machine-learning classification. We describe the architecture, the configuration of each component, the containment measures, and the validation procedure, and we discuss the limits of a simulation-driven evaluation. The work shows that a complete, safely contained monitoring environment can be assembled from open-source components on modest hardware and used as a repeatable testbed for security education and for research on attack classification.**

***Index Terms*—honeypot, deception technology, network security monitoring, intrusion detection, T-Pot, Cowrie, Suricata, ELK stack, attack simulation.**

I. INTRODUCTION

Organisations today defend networks against a wide range of threats, from automated scanners that sweep the address space for exposed services to patient, targeted intrusions. Perimeter devices, antivirus products and signature-based intrusion detection systems (IDS) remain necessary, but they share a structural weakness: they react to patterns that someone has already seen. An attacker who uses a new exploit, a modified tool or stolen but valid credentials can pass through such controls unnoticed, and the defender frequently learns of the intrusion only after data has been lost. The imbalance is easy to state: defenders have to secure every entry point, whereas an attacker needs only one.

Deception technology tries to reverse this imbalance. A *honeypot* is a system that is deliberately exposed and instrumented so that it can be probed, attacked and even compromised while the defender records everything that happens [3]. Since no genuine business function depends on the decoy, any connection to it is unsolicited by definition. This property gives honeypots an unusually favourable signal-to-noise ratio: the alerts they produce are rarely false positives, and the captured sessions contain credentials, commands, payloads and tooling that reveal what the adversary intends to do.

Turning a honeypot into a monitoring capability requires more than starting a decoy service. Events have to be collected from several sensors, normalised, enriched with context such as geolocation, correlated with IDS alerts and presented to an analyst in a form that supports decisions. The decoys themselves must be isolated so that a successful attacker cannot use them as a stepping stone. Many student and small-organisation projects stop at the first step and never address these integration and safety questions.

The contributions of this paper are as follows.

1) We describe a layered architecture that unites several low- and medium-interaction decoys, a passive IDS and a centralised log-analytics pipeline in one isolated virtual machine.
2) We document the configuration of the platform and a set of containment measures that prevent the decoy from being abused for outbound attacks or lateral movement.
3) We present Python traffic simulators and a web-based Control Center that make the environment repeatable: attack and benign sessions can be generated on demand, inspected, filtered and exported as a labelled dataset.
4) We outline a validation procedure based on controlled attack tools and discuss candidly what a simulation-driven evaluation can and cannot establish.

The remainder of the paper is organised as follows. Section II reviews related work. Section III gives background on honeypot classes and the T-Pot components. Section IV presents the architecture, and Section V the



implementation. Section VI describes the simulators and the Control Center. Section VII reports the validation and discusses the findings. Section VIII covers legal and ethical aspects, Section IX lists limitations and future work, and Section X concludes.

II. PROBLEM AND OUTCOMES

Raw honeypot logs are often difficult to interpret, especially when events are produced by multiple sensors in different formats. Manual review makes it harder to identify repeated login failures, scan patterns, and the relationship between a network alert and a recorded session. In addition, testing based only on live attackers is unpredictable and cannot be repeated consistently. The problem addressed is therefore the lack of an integrated, safely contained, and repeatable monitoring workflow that converts decoy activity into information an administrator can interpret.

The proposed system aims to: (1) capture suspicious connections and session details; (2) identify brute-force and reconnaissance behaviour using rules and a detection pipeline; (3) correlate decoy records with passive IDS alerts; (4) present event summaries and session details through a dashboard; (5) generate repeatable labelled traffic for evaluation and future machine-learning experiments; and (6) contain the decoy environment so it cannot be used to reach production systems or attack third parties

Other researchers studied the trade-off between realism and risk. Alata et al. deployed a high-interaction honeypot with real services exposed and analysed the commands, tools and behaviour of attackers who gained access, reporting lessons about both attacker habits and the operational difficulties of running such a system [7]. High-interaction systems yield detailed behavioural evidence, but they require careful containment because the attacker is interacting with a real operating system. Low- and medium-interaction systems emulate services in software and are easier to secure, at the cost of shallower data and a greater chance of being fingerprinted.

More recent work surveys the software landscape and the problems of analysing honeypot data. Nawrocki et al. compare a large number of honeypot implementations, classify them by the services they emulate and the way they are deployed, and discuss data-analysis and sharing practices as well as the difficulty of detecting large-scale activity in cloud settings [8]. Their survey also highlights that integrated distributions, which combine several decoys with a shared analytics stack, lower the barrier to adoption. The T-Pot platform used in the present work is one such distribution [9].

In comparison with this literature, the present paper does not claim a new detection algorithm. Its aim is to show how established components can be integrated into a contained, observable and repeatable environment, and to add a simulation and exploration layer that supports teaching and the preparation of labelled datasets for machine-learning experiments.

III. LITERATURE SURVEY

A. Early work on deception and honeypots

Stoll's account of tracing an intruder through a university network illustrated the investigative value of observing an attacker over time [1]. Cohen later discussed deception as a component of information protection, including the use of false systems and information to reveal unauthorised behaviour [2]. Spitzner consolidated practical guidance on honeypot design, deployment, and operational risks, and distinguished production honeypots from research honeypots [3]. These works establish the central idea used in this project: a decoy can provide a concentrated record of activity that would otherwise be difficult to observe.

B. Virtual and distributed monitoring

Provos presented a virtual honeypot framework that enabled multiple simulated hosts to be monitored using a single physical machine [4]. Provos and Holz subsequently described virtual honeypots, malware collection, botnet tracking, and the use of honeypot data for intrusion detection [5]. Bailey et al. developed the Internet Motion Sensor, a distributed blackhole monitoring system that observed traffic directed at unused address space and helped characterise internet-wide events [6]. This line of research demonstrates the value of automation and distributed observation, while also motivating centralised collection and analysis.

C. Interaction level and operational trade-offs

Alata et al. reported lessons from deploying a high-interaction honeypot, showing that realistic environments can reveal detailed attacker commands and tools but require careful containment and maintenance [7]. Low-interaction honeypots are easier to operate and expose less of the host, but collect less behavioural detail. Medium-interaction systems emulate enough service behaviour to capture commands and sessions while avoiding direct exposure of a production operating system. The present design uses this practical balance through decoy services such as Cowrie and other T-Pot components.

D. Modern honeypot software and integrated analytics

Nawrocki et al. surveyed honeypot software and data-analysis approaches, comparing deployment methods, service emulation, and challenges in collecting and analysing large-scale data [8]. Their survey supports the use of integrated platforms rather than isolated tools. T-Pot packages multiple honeypots with network monitoring and visual analytics [9]. Cowrie records SSH and Telnet login attempts and shell activity [10], while Suricata provides network IDS/monitoring



reason for choosing a distribution rather than assembling the tools by hand.

IV. SYSTEM ARCHITECTURE

The system is organised in five layers, shown in Fig. 1. Traffic from real or simulated attackers first meets the honeypot layer, in which each decoy listens on the ports normally associated with the service it imitates. A passive instance of Suricata observes the same network interface and raises alerts when traffic matches its rule set. All components write their records to a shared volume in JSON format. The collection layer, made up of Filebeat and Logstash, reads these files, parses them into fields and adds geolocation information before indexing the events in Elasticsearch. The presentation layer consists of Kibana dashboards, the machine-learning models, and the custom Control Center that sits beside them.

Three design decisions shaped the architecture. First, the honeypot machine is placed on an isolated network segment with no route to production resources, so that compromise of a decoy cannot spread. Second, the administrative interface is separated from the decoys: the real SSH daemon is moved to a high port so that the decoy can occupy the standard port 22, and management is reachable only with key-based authentication from allow-listed addresses. Third, detection is deliberately passive. Suricata does not drop packets; its purpose is to annotate traffic, so that an analyst can compare what the IDS recognised with what the decoys recorded.

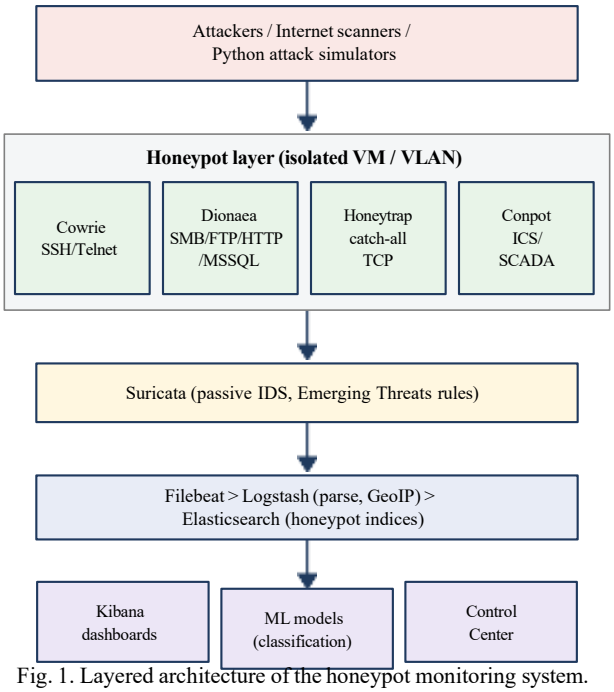


Fig. 1. Layered architecture of the honeypot monitoring system.

V. IMPLEMENTATION

A. Environment

The host is a virtual machine with four virtual CPUs, between 8 and 16 GB of memory and 128 GB of storage running Ubuntu Server 22.04. The machine is attached to a host-only network or a dedicated VLAN behind a controlled gateway. These resources are sufficient because the decoys are lightweight; the Elasticsearch service is the dominant consumer of memory and disk.

B. Installing the Platform

After updating the package index and installing git and curl, the T-Pot repository is cloned and its installer is executed. The installer offers several installation types, and we selected the standard type, which starts the full set of decoys together with the analytics stack. An administrative web user is created during installation and the machine is rebooted; the containers then start automatically and the web interface becomes available over HTTPS on the configured management port.

```
sudo apt update && sudo apt install -y git curl
git clone https://github.com/telekom-security/tpotce
cd tpotce && ./install.sh   # choose Standard type
sudo reboot
```

C. Decoy Configuration

Cowrie listens on ports 22 and 23 and presents an emulated Linux shell with a fake file system. It is configured to accept a number of weak passwords, since an attacker who is refused every attempt leaves no post-login behaviour to analyse, and it logs the complete session, including every command typed and every file fetched. Dionaea listens on the SMB, FTP, HTTP and MSSQL ports and is intended to capture exploitation attempts and the malware that follows. Honeytrap registers a catch-all listener for TCP ports not claimed by another decoy, which reveals scanning of unusual services. Conpot simulates industrial control devices so that probes against SCADA protocols become visible. Suricata runs in passive mode on the honeypot interface with the Emerging Threats rule set kept up to date.

D. Log Collection and Enrichment

Every decoy writes JSON records to a volume that is shared with the collection containers. Filebeat reads the files and forwards them to Logstash, which extracts the timestamp, source address, destination port, username, password, command and file hash, and adds geolocation data derived from the source address. The events are indexed in Elasticsearch under indices specific to each honeypot. Hashes of captured files can be checked against the VirusTotal service to find out whether a sample is already known, which separates commodity malware from material that deserves closer study.

E. Security and Containment

A honeypot that can be turned into an attack platform is a liability, so the following measures were applied. Outbound traffic from the honeypot is blocked or rate-limited, which



prevents it from sending spam or participating in denial-of-service campaigns. There is no route from the honeypot segment to production networks. The containers run with limited privileges, and periodic snapshots of the virtual machine allow rapid restoration to a known-clean state after a compromise. Administrative access is restricted by key-based authentication and an IP allow-list.

TABLE IV
CONTAINMENT CONTROLS AND THE RISK EACH ADDRESSES

Control	Risk addressed
Outbound traffic blocked or rate-limited	Decoy used to send spam or join denial-of-service attacks
No route to production network	Lateral movement after a compromised decoy
Limited container privileges	Container escape to the host
Periodic VM snapshots	Long recovery time after a compromise
Key-based SSH, IP allow-list, moved admin port	Attacker reaching the management interface

F. Pre-Deployment Testing

Before the system is considered ready, a separate attacker machine is used to exercise it. A TCP SYN scan with service detection across all ports checks that the decoys respond as expected, and a dictionary-based SSH password attack checks that credential attempts reach Cowrie.

```
nmap -sS -sV -p- <honeypot-ip>
hydra -l root -P passwords.txt ssh://<honeypot-ip>
```

The test is passed when each attempt is visible within a few seconds in the raw honeypot logs, in the Suricata alerts and in the Kibana dashboard. This end-to-end check matters more than any single component test, because a monitoring system is only as good as the slowest or weakest link in the path from the wire to the analyst.

VI. RELATED WORK

The idea of luring and observing an intruder predates the word honeypot. Stoll's account of tracing a foreign intruder through a university network in the late 1980s showed how much can be learned by watching an attacker who believes he is unobserved [1]. Cohen subsequently examined deception as a systematic element of information protection, arguing that false systems and false information can be used to detect and study unauthorised behaviour [2]. Spitzner's book consolidated these ideas into practical guidance on the design, deployment and legal handling of honeypots and introduced the distinction between production and research honeypots that is still widely used [3].

A second line of work addressed scale and automation. Provos described a framework that simulates many virtual hosts on unused address space, which made it practical to monitor large networks with a single physical machine [4]. The later monograph by Provos and Holz covered virtual honeypots, malware collection and the use of honeypot data for botnet tracking and intrusion detection [5].

brute-force session repeatedly tries credentials against the login service. A *reconnaissance* session probes the service and its surroundings without attempting to log in repeatedly. A *normal-user* session imitates a benign login and a few ordinary commands. Including benign sessions is important: a classifier that has seen only attacks cannot be assessed for its false-positive behaviour.

TABLE II
SIMULATED SESSION CLASSES

Class	Intended behaviour	Expected trace in logs
Brute force	Many credential guesses in quick succession	Repeated failed logins from one source; short sessions
Reconnaissance	Probing of ports and service banners	Connections with little or no authentication; scan alerts
Normal user	Occasional, benign interaction	Single login, few commands, ordinary session length

A. Control Center

The Honeypot Control Center is a browser-based dashboard that gives operators a single place to manage the environment. Its landing page reports the readiness of the application programming interface, the state of the machine-learning models, the availability of the dataset, the status of the honeypot and whether the process holds administrator privileges. From the same page an operator can launch an attack simulation, open an explorer for the captured sessions, filter them by type, and export the dataset. When the honeypot is not running the interface says so and warns that a simulation request would be rejected, which prevents silent failure and makes the dependencies between components visible.

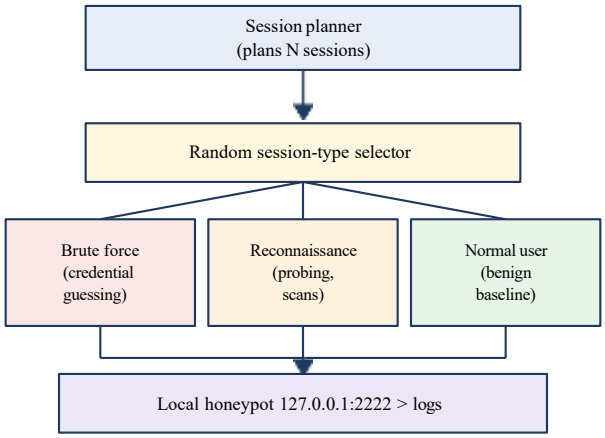


Fig. 2. Operation of the automated traffic simulator.

B. Dataset Preparation for Machine Learning

Because the simulators know which behaviour they produced, each generated session carries a ground-truth label. The export function therefore yields a labelled dataset that can be used to train and test classifiers that separate brute-force, reconnaissance and benign sessions from features such as the number of login attempts, session



duration and command count. The repository includes a models directory and a data-generation script for this purpose. We stress that labels obtained from a simulator describe the simulator, not the world; models trained on them must be re-validated on traffic captured from real attackers before any operational use.

VII. RESULTAND AND DISCUSSION

A. What Was Resultand

The resultand in this phase focuses on functional correctness of the monitoring path rather than on large-scale statistics. Three questions were examined. First, does every probe from the attacker machine appear in the decoy logs, the IDS alerts and the dashboard promptly? The scanning and password-guessing tests described in Section V-F were used for this purpose and each attempt was observed in all three places within seconds. Second, can the simulators drive the honeypot with a mixture of session types without manual intervention? The two runs, with 186 and 194 planned sessions, completed in this way, and the terminal output enumerates each session and its type as it executes. Third, does the Control Center expose the system state and the captured data accurately? The dashboard displayed the component status and the simulation and data-exploration options as designed.

B. Comparison with Signature-Based Monitoring

Table III contrasts the monitoring properties of the honeypot approach with those of a conventional signature-based IDS on the three limitations that motivated this work.

TABLE III
COMPARISON OF MONITORING APPROACHES

Property	Signature IDS	Honeypot monitoring
False positives	Can be frequent on busy networks	Rare; no legitimate traffic expected
Unknown attacks	Missed without a matching rule	Recorded because the decoy is contacted
Behavioural context	Usually an alert only	Credentials, commands and files captured
Coverage	Traffic that crosses the sensor	Only traffic aimed at the decoy
Operational risk	Low	Needs strict containment

The comparison also shows why the two technologies complement one another. A honeypot sees nothing of traffic that never touches it, so it cannot replace a network IDS; conversely, the IDS cannot explain what an attacker did after a successful login. In the deployment described here the passive Suricata instance supplies the signature view while the decoys supply the behavioural view, and the shared index allows both to be read side by side.

C. Observations

Several practical lessons emerged. Moving the real administrative SSH service to a high port before installation avoids a conflict that would otherwise prevent Cowrie from binding to port 22. Allowing weak passwords increases the amount of post-login activity, but it also means that the quality of the fake file system determines how long an attacker stays interested. The Elasticsearch component requires far more memory than the decoys, so the lower end of the 8–16 GB range is only adequate for light workloads. Finally, a visible status panel in the Control Center reduced time lost to start-up errors, since the operator can see immediately whether the honeypot or the model service is the missing piece.

D. Threats to Validity

Simulated traffic is generated by code that the authors wrote, so it is considerably more regular than the behaviour of real adversaries, who vary their tools, timing and persistence. Results obtained from it demonstrate that the pipeline functions; they do not measure how well the system would perform against the diversity of internet attacks. In addition, the local target address used in the simulation means that network-level effects such as latency, packet loss and source-address diversity are absent. These limits are inherent to the chosen evaluation strategy and are addressed in the plan for future work.

VIII. LEGAL AND ETHICAL CONSIDERATIONS

Operating a honeypot raises legal and ethical questions that should be settled before any system is connected to a public network. The first is liability: if a decoy is used to attack a third party, the operator may be held responsible, which is why outbound filtering is regarded as mandatory rather than optional [3]. The second concerns privacy and interception law, because honeypots record the activity of whoever connects, and some jurisdictions regulate the collection of such data and the retention of address information. The third is the matter of entrapment, which in most settings applies to law-enforcement action and not to passive research decoys, but which is nevertheless a reason to document the purpose of the system. All experiments reported here were performed on a local, isolated testbed using attack tools run by the authors against their own machines, and no third-party data were collected.

IX. LIMITATIONS AND FUTURE WORK

The present work has the following limitations. The evaluation relies on simulated sessions against a local honeypot and has not yet been exposed to sustained internet traffic. The decoys are low- and medium-interaction, which restricts the depth of attacker behaviour that can be observed and leaves them open to fingerprinting by sophisticated adversaries. The machine-learning component is at the stage of dataset preparation and model scaffolding, and its accuracy has not yet been established on independent data.

Several extensions are planned. The first is a staged internet-facing deployment on a cloud instance with the same containment controls, so that real attack data can be gathered over an extended period and used to measure session volume, source diversity and attack categories. The



second is to map observed command sequences to the MITRE ATT&CK framework [11], which would convert raw logs into statements about tactics and techniques. The third is to train and evaluate classifiers on a mixture of simulated and captured sessions, using cross-validation and reporting precision, recall and false-positive rate. The fourth is automated alert delivery through e-mail or short messages and the generation of periodic summary reports. A longer-term direction is adaptive deception, in which the decoy configuration is changed automatically in response to observed attacker behaviour.

X. CONCLUSION

This paper described the design and implementation of a honeypot-based network security monitoring system assembled from the open-source T-Pot platform, and extended with Python traffic simulators and a web-based Control Center. The architecture integrates several decoys, a passive IDS and a centralised analytics pipeline inside an isolated virtual machine, and its containment measures address the main operational risk of running deliberately exposed systems. The simulators make the environment repeatable and produce labelled sessions that can support machine-learning experiments, while the validation procedure confirms that attacker activity is visible in logs, alerts and dashboards within seconds. Honeypots do not replace conventional defences, but they add a low-noise source of behavioural evidence that signature-based tools cannot provide. The next stage of the project will expose the system to real traffic and quantify its detection and classification performance.

ACKNOWLEDGMENT

The authors thank the management, the Principal and the Head of the Department of Computer Science and Engineering of Akshaya Institute of Technology, Tumakuru, for providing laboratory facilities, and Prof. Thippamma S for her guidance throughout the project.

REFERENCES

[1] C. Stoll, *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. New York, NY, USA: Doubleday, 1989.

[2] F. Cohen, "A note on the role of deception in information protection," *Computers & Security*, vol. 17, no. 6, pp. 483–506, 1998.

[3] L. Spitzner, *Honeypots: Tracking Hackers*. Boston, MA, USA: Addison-Wesley, 2002.

[4] N. Provos, "A virtual honeypot framework," in *Proc. 13th USENIX Security Symp.*, San Diego, CA, USA, 2004, pp. 1–14.

[5] N. Provos and T. Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Upper Saddle River, NJ, USA: Addison-Wesley, 2007.

[6] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, "The Internet Motion Sensor: A distributed blackhole monitoring system," in *Proc. Network and Distributed System Security Symp. (NDSS)*, San Diego, CA, USA, 2005.

[7] E. Alata, V. Nicomette, M. Kaâniche, M. Dacier, and M. Herrb, "Lessons learned from the deployment of a high-interaction honeypot," in *Proc. 6th European Dependable Computing Conf. (EDCC)*, Coimbra, Portugal, 2006, pp. 39–46.

[8] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A survey on honeypot software and data analysis," arXiv:1608.06249, 2016.

[9] Telekom Security, "T-Pot: The all in one honeypot platform." GitHub repository. [Online]. Available: https://github.com/telekom-security/tpotce

[10] M. Oosterhof, "Cowrie SSH/Telnet honeypot." GitHub repository. [Online]. Available: https://github.com/cowrie/cowrie

[11] B. E. Strom *et al.*, "MITRE ATT&CK: Design and philosophy," The MITRE Corporation, McLean, VA, USA, Tech. Rep., 2020.

[12] Open Information Security Foundation, "Suricata: Network IDS, IPS and network security monitoring engine." [Online]. Available: https://suricata.io