Enabling AI-Driven Cybersecurity Through Reproducible Cloud-Native Security Architectures

Gianni D' Angelo(✉), Carmine Ambrosino, Simona Ettari, Alfonso Petrosino, and Francesco Palmieri

Department of Computer Science, University of Salerno, Fisciano, Salerno, Italy
{giadangelo,fpalmieri}@unisa.it,
{c.ambrosino11,s.ettari,a.petrosino39}@studenti.unisa.it

Abstract. Artificial intelligence is increasingly used in cybersecurity for anomaly detection, behavioral modeling, and automated defense. However, the reproducibility of AI-based cybersecurity experiments depends not only on model code and datasets, but also on the infrastructure used to generate security data. In cloud-native environments, reverse proxies, storage backends, monitoring pipelines, timeout policies, and network configurations can shape traffic behavior and introduce hidden biases into detection models. This paper presents *local-lab*, an Infrastructure as Code-based architecture for building isolated and repeatable cloud-native security testbeds. The platform uses OpenTofu and `libvirt` to instantiate a stack composed of Nextcloud, a Caddy reverse proxy, MinIO object storage, and a monitoring pipeline based on Netdata, Prometheus, and Grafana. We use this environment to execute controlled Slow HTTP Denial-of-Service attacks, collect packet-level and system-level telemetry, construct the labeled SLOWDOS-2025 dataset, and evaluate neural detection models, including multilayer perceptrons, long short-term memory networks, and autoencoder-based configurations. The results show that the proposed infrastructure supports controlled observation of SlowDoS behavior across multiple layers and provides a structured setting for evaluating AI-based detection models. Beyond classification performance, the main contribution of this work is to treat the experimental infrastructure itself as a version-controlled scientific artifact, improving transparency, repeatability, and comparability in AI-driven cybersecurity research.

Keywords: Cloud-native security · Reproducibility · Local lab · Intrusion detection · Slow DoS attacks · Cybersecurity dataset

1 Introduction

Artificial Intelligence (AI) is widely used in cybersecurity for anomaly detection, behavioral modeling, and automated defense. Yet the reliability of AI-based

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16759, pp. 575–591, 2027.
https://doi.org/10.1007/978-3-032-30518-3_36

cybersecurity experiments depends not only on learning algorithms, model code, or released datasets, but also on the infrastructure in which security data are produced. In cloud-native environments, components such as reverse proxies, object storage backends, monitoring services, timeout policies, and network configurations can directly affect traffic behavior. Even small infrastructure changes may therefore alter the generated data and influence the evaluation of detection models. This raises a reproducibility problem. Many cybersecurity studies release datasets or source code, while the experimental infrastructure that generated the data is only partially described, manually configured, or left implicit. As a result, the environment becomes a hidden experimental variable. Differences in service deployment, connection handling, storage integration, and telemetry collection can introduce biases that are difficult to identify and make results harder to compare across studies. This paper investigates how cloud-native security experiments can be made more transparent and repeatable by treating the infrastructure itself as a scientific artifact. We present *local-lab*, an Infrastructure as Code-based architecture that enables consistent deployment and reconstruction of an isolated cloud-native stack composed of Nextcloud, a Caddy reverse proxy, MinIO object storage, and a monitoring pipeline based on Netdata, Prometheus, and Grafana. We evaluate the architecture through Slow HTTP Denial-of-Service (SlowDoS) experiments. SlowDoS attacks are well suited to this setting because they exploit timing, connection persistence, and resource management rather than high traffic volume. Their behavior is therefore strongly influenced by infrastructure-level factors such as proxy timeout policies, connection handling, and service configuration. In our experiments, benign and malicious traffic are collected from the deployed platform, transformed into flow-based features, and used to evaluate neural detection models, including multilayer perceptrons, long short-term memory networks, and autoencoder-based representations.

The study is guided by the following research questions:

- **RQ1:** Can cloud-native security architectures be formally specified and instantiated as version-controlled artifacts for reproducible cybersecurity experiments?
- **RQ2:** Can such an architecture support controlled observation of application-layer SlowDoS attacks across multiple infrastructure layers?
- **RQ3:** Can it provide a structured experimental environment for evaluating AI-based SlowDoS detection models?

The paper differs from prior work on cyber ranges, SlowDoS detection, stealthy cloud denial-of-service strategies, and machine learning-based intrusion detection. Its primary goal is not to introduce a new attack strategy, training scenario, or classifier. Instead, it focuses on the infrastructure conditions under which security data are generated and models are evaluated. By specifying the complete experimental environment as version-controlled Infrastructure as Code artifacts, the proposed approach makes the data-generation process more transparent, repeatable, and easier to inspect.

The main contributions of this paper are as follows:

1. We define a cloud-native security architecture based on Infrastructure as Code, where virtual machines, network topology, service deployment, monitoring components, and attack execution are specified as version-controlled artifacts.
2. We instantiate the architecture through the open-source *local-lab* project and use it to execute controlled SlowDoS experiments against a realistic Nextcloud-based application stack integrating Caddy and MinIO.
3. We generate the SLOWDOS-2025 dataset from packet captures collected during benign and malicious scenarios, using a documented pipeline for flow-based feature extraction, labeling, preprocessing, and balancing.
4. We evaluate multiple neural models, including MLP, LSTM, and autoencoder-based configurations, showing how the proposed environment supports AI-based SlowDoS detection studies under controlled experimental conditions.

The remainder of the paper is organized as follows. Section 2 discusses related work. Sections 3 and 4 describe the methodology and architecture. Section 5 presents the experimental results, limitations, and threats to validity. Section 6 concludes the paper.

2 Related Work

This work builds on several research areas, including cyber range infrastructures, low-rate and Slow HTTP denial-of-service attacks, machine learning-based intrusion detection, and dataset generation for security experimentation. Although these research directions have evolved significantly, existing studies rarely integrate reproducible infrastructure deployment, controlled attack execution, telemetry collection, dataset generation, and machine learning evaluation into a single end-to-end workflow.

2.1 Cyber Range Infrastructures and Reproducible Security Testbeds

Cyber ranges have emerged as important instruments for cybersecurity experimentation, training, and evaluation. Surveys of cyber range platforms highlight significant heterogeneity in architecture, deployment strategies, and experimental objectives [25]. Modern platforms increasingly rely on virtualization and automated orchestration to support dynamic scenario creation and infrastructure scalability [26]. Container-based approaches further improve portability and reproducibility by enabling lightweight deployment of complex service architectures through container orchestration frameworks [27]. Recent work has explored model-driven and automatically generated cyber ranges capable of synthesizing realistic infrastructures from high-level descriptions of systems and attack scenarios [28]. Other studies have investigated scalable cyber range architectures based on cloud technologies, enabling the deployment of large-scale experimental environments for cybersecurity research [29]. Despite these advances, most

cyber range systems are primarily designed for training, education, or red–blue team exercises. In contrast, our work treats the experimental infrastructure itself as a scientific artifact whose configuration can be versioned, reconstructed, and reused for machine learning-based cybersecurity experimentation.

2.2 Low-Rate and Slow HTTP Denial-of-Service Attacks

Denial-of-service attacks remain one of the most widely studied threats affecting networked systems. Low-rate denial-of-service attacks are particularly challenging because they exploit protocol dynamics and server resource management rather than high traffic volumes. Early work on low-rate TCP-targeted attacks demonstrated that carefully crafted periodic traffic patterns can degrade service availability while remaining difficult to detect through conventional monitoring mechanisms [30]. Slow HTTP attacks constitute a relevant subclass of these techniques. Slowloris maintains large numbers of partially completed HTTP requests that force servers to keep connections open indefinitely [31]. Similarly, slow HTTP POST attacks send request bodies at extremely low transmission rates, exhausting server-side connection pools. Surveys on low-rate denial-of-service detection highlight the difficulty of identifying these attacks because of their low traffic footprint and similarity to legitimate client behavior [32]. Experimental studies further show that SlowDoS effectiveness depends on infrastructure configuration, including reverse proxies, load balancers, and web application gateways [33]. Stealthy denial-of-service strategies in cloud computing have also been studied from the perspective of resource exhaustion under low-profile traffic patterns. Ficco and Rak propose a strategy for orchestrating slowly increasing attack patterns against cloud applications, with the objective of degrading service and increasing resource consumption while delaying detection [42]. This line of work is closely related to SlowDoS attacks because both exploit the interaction between application-layer behavior, resource allocation, and infrastructure configuration. Our work differs in its primary objective. Rather than proposing a new stealthy attack strategy or a dedicated detection algorithm, it focuses on the reproducibility of the infrastructure used to generate and evaluate security data. By combining Infrastructure as Code deployment, multi-layer telemetry collection, packet capture, dataset generation, and machine learning evaluation, the proposed framework enables SlowDoS behavior to be analyzed as an infrastructure-dependent phenomenon within a reproducible experimental workflow.

2.3 Machine Learning for Intrusion Detection

Machine learning techniques have been widely adopted for intrusion detection systems. Traditional IDS research initially relied on classical algorithms such as decision trees, support vector machines, and clustering-based anomaly detection [34]. More recently, deep learning models have shown strong potential

for detecting complex attack patterns in network traffic data. Recurrent neural networks and Long Short-Term Memory architectures can capture temporal dependencies in network flows [35], while autoencoders have been used for anomaly detection by learning compact latent representations of normal traffic behavior [36]. Several studies have specifically investigated deep learning approaches for detecting slow HTTP denial-of-service attacks. Flow-based classification models using statistical traffic features have shown promising results in distinguishing slow HTTP attacks from legitimate traffic [37]. Other work has explored feature selection and hybrid machine learning techniques for improving detection performance [38]. However, many of these studies rely on datasets whose generation processes are not fully documented, making it difficult to assess whether classification performance is influenced by the model itself or by characteristics of the dataset generation environment.

2.4 Dataset Generation and Feature Extraction for IDS Research

The reliability of machine learning-based intrusion detection depends heavily on the quality and realism of the datasets used during training and evaluation. Widely used benchmark datasets have been shown to contain limitations such as outdated traffic patterns, unrealistic attack scenarios, or incomplete documentation of the experimental setup used during data collection [39]. Recent work has therefore focused on improving dataset generation methodologies and traffic feature extraction pipelines. Flow analysis frameworks such as NTLFlowLyzer enable the extraction of statistical features from packet captures, supporting expressive representations of network traffic [23]. Other work has explored realistic network dataset generation through controlled cyber range environments [40]. Nevertheless, realistic IDS dataset generation remains challenging because it requires experimental environments capable of orchestrating complex service infrastructures, generating representative benign workloads, executing controlled attack scenarios, and collecting telemetry across multiple infrastructure layers.

2.5 Research Gap and Relation to the Research Questions

Existing research has addressed cyber range infrastructures, SlowDoS detection, stealthy cloud DoS strategies, machine learning-based intrusion detection, and dataset generation largely as separate research directions. Cyber range platforms mainly focus on training or red–blue team exercises. SlowDoS and stealthy DoS studies often analyze attack behavior or detection strategies without providing a sufficiently specified experimental infrastructure. Machine learning-based IDS studies frequently evaluate models on datasets whose generation process is only partially specified. This fragmentation creates a gap in experimental cybersecurity research: the lack of frameworks that jointly support infrastructure deployment, controlled SlowDoS experimentation, telemetry collection, dataset generation, and machine learning evaluation in a repeatable workflow. The proposed

Table 1. Qualitative positioning of related work with respect to the reproducible experimental pipeline addressed in this paper.

Reference	Infra.	Stack	SlowDoS	ML
Munaiah et al. [25]	P	–	–	–
Yamin et al. [26]	P	–	–	–
Nakata and Otsuka [27]	✓	P	–	–
Ficco and Rak [42]	–	P	✓	–
Aladaileh et al. [37]	–	–	✓	✓
Sharafaldin et al. [40]	P	P	P	✓
This work	✓	✓	✓	✓

Legend. A check mark (✓) indicates that the dimension is explicitly implemented and represents a central part of the work. The symbol *P* indicates partial support: the dimension is discussed, indirectly supported, or only partially implemented, but not specified in enough detail to allow direct reconstruction in the context considered here. A dash (–) indicates that the dimension is not addressed.
Dimensions. *Infra.* refers to an explicitly specified and reproducible experimental infrastructure, such as deployment artifacts, configuration files, or sufficiently detailed setup descriptions. *Stack* refers to a realistic multi-component application stack, rather than a single isolated service or a purely synthetic scenario. *SlowDoS* refers to explicit support for Slow HTTP, low-rate, or stealthy denial-of-service experimentation. *ML* refers to the training or evaluation of machine learning models on the generated or analyzed data. Assignments were made conservatively, based only on aspects explicitly described in the cited papers.

local-lab architecture addresses this gap by treating the experimental infrastructure as a version-controlled scientific artifact. Compared with existing cyber range platforms, the proposed work places less emphasis on training scenarios and greater emphasis on reproducible experimentation. Compared with prior SlowDoS and stealthy DoS studies, it analyzes attack behavior while explicitly specifying the infrastructure conditions under which traffic is generated. Compared with ML-based IDS studies, the contribution is not a new classifier, but a controlled and reproducible data-generation and evaluation pipeline. These aspects directly support RQ1, RQ2, and RQ3.

To improve transparency, Table 1 reports a qualitative positioning of related work according to four dimensions: reproducible infrastructure, realistic application stack, SlowDoS experimentation, and machine learning evaluation. The legend and assignment criteria are reported in the table notes. The table is therefore intended as a qualitative positioning of research scope, not as a ranking of scientific quality or effectiveness.

Overall, Table 1 shows that existing works usually address only part of the experimental pipeline. Cyber range studies emphasize deployment and training scenarios, SlowDoS studies focus on attack behavior or detection, and IDS dataset works mainly address data generation or model evaluation. In contrast, the proposed architecture integrates reproducible infrastructure deployment, controlled SlowDoS execution, telemetry collection, dataset construction, and neural model evaluation within a single reproducible workflow.

3 Materials and Methods

3.1 Reproducible Experimental Infrastructure

The experimental evaluation was conducted in a fully virtualized and isolated cybersecurity testbed instantiated through an Infrastructure as Code (IaC) workflow. This approach supports controlled experimentation by allowing the entire infrastructure configuration to be automatically deployed and consistently reconstructed. OpenTofu [8] orchestrates provisioning through the `libvirt` provider in a NAT-based private network (`10.0.2.0/24`). The infrastructure reproduces a simplified cloud-native deployment scenario while keeping the experimental environment fully contained.

Our `local-lab` includes three nodes:

1. **Target node:** hosts the application stack composed of Nextcloud behind a Caddy [1] reverse proxy with MinIO [2] as S3-compatible object storage.
2. **Monitoring node:** collects telemetry and system metrics.
3. **Attacker node:** generates malicious traffic for the attack scenarios.

This architecture enables simultaneous observation of application behavior, network traffic, and system-level resource consumption.

3.2 Infrastructure as Code for Reproducibility

Traditional cybersecurity testbeds are often assembled manually, for example by creating virtual machines, configuring networks, installing services, and applying security settings step by step. This process can introduce configuration drift and limit portability across research settings. To mitigate these issues, the proposed architecture adopts an Infrastructure as Code workflow in which topology, VM resources, networking, and service composition are explicitly described and version-controlled. The declarative OpenTofu-based workflow enables automated deployment and consistent reconstruction of the experimental platform.

3.3 Attack Scenarios and Execution Methodology

To evaluate the cloud-native service stack under resource exhaustion, SlowDoS attacks were simulated after deployment. Two tools were used:

1. Slowloris, which maintains multiple incomplete HTTP header requests to saturate server sockets.
2. slowhttptest [22], which generates Slow POST attacks by transmitting HTTP body data at very low rates.

The attack execution follows a workflow inspired by the Penetration Testing Execution Standard (PTES) [41], enabling controlled and systematic generation of malicious traffic.

3.4 Telemetry Collection and Traffic Capture

Two complementary telemetry channels were collected during the experiments:

1. **System and service monitoring:** Netdata collected fine-grained indicators, including TCP sockets, open connections, CPU usage, and memory usage, which were periodically scraped by Prometheus and visualized through Grafana dashboards.
2. **Packet capture:** packet-level traffic was captured on the target node using Wireshark or `tcpdump` under three operational conditions: benign usage, Slowloris, and Slow POST. These captures were used to generate the dataset for the machine learning analysis.

3.5 Dataset Construction

Packet captures were converted into flow-based tabular datasets using NTLFlowLyzer [23], which extracts statistical features from network flows. For each PCAP file, a configuration specifies the input file, output dataset, traffic label (Normal or SlowDoS), and processing parameters. The resulting labeled CSV files were merged into a unified dataset. Duplicate flows were removed, and balanced instances of benign and malicious traffic were combined to build the final dataset, SLOWDOS-2025.

3.6 Preprocessing and Transformation

The dataset contained missing and infinite values across multiple features.

Columns with more than 15% missing values and low relevance were removed, while the remaining missing values were imputed using the median. Infinite values were replaced with the median computed after excluding infinities. Moreover, the presence of `"not a complete handshake"` values in handshake-related features was leveraged through feature engineering by introducing a boolean indicator, `is_complete_handshake`. The preprocessed dataset was exported as `slowdos_2025_preprocessing.csv`.

A second dataset, `slowdos_2025_mix_preprocessing.csv`, was generated to evaluate a reduced feature subset inspired by prior work on slow DoS feature behavior. Finally, numerical features were standardized with `StandardScaler`, producing `std_slowdos_2025_preprocessing.csv`.

3.7 Neural Models

Three neural approaches were evaluated for binary classification (Normal vs. SlowDoS): an autoencoder, two MLP classifiers, and an LSTM sequence model. The autoencoder compresses the input feature space into latent representations of size 128, 64, or 16, which are subsequently used as inputs for supervised classifiers. Two MLP architectures were implemented: a compact baseline with 64–32 neurons and a deeper architecture with 128–64–32 neurons, both using ReLU activation, batch normalization, and dropout. To capture temporal dependencies in traffic flows, the LSTM model processes sequences constructed through a sliding window of length 50 and step 5. The architecture includes an LSTM layer with 32 units followed by dense layers and a sigmoid output. All models were implemented in TensorFlow/Keras and trained using the Adam optimizer.

3.8 Evaluation Protocol and Metrics

The experimental workflow integrates automated architecture deployment, SlowDoS attack execution, telemetry collection, and machine learning analysis within a reproducible cloud-native environment. Packet captures collected during the experiments are converted into flow-based datasets and preprocessed before being used to train and evaluate neural models under multiple feature configurations, including the full feature set, the FeatureMix subset, and autoencoder embeddings. To avoid potential data leakage, the train, validation, and test partitions were defined at the capture-session level, using PCAP files as the splitting unit. In particular, all flows extracted from the same PCAP capture were assigned to the same partition before constructing LSTM sliding windows. This protocol prevents overlapping or near-duplicate temporal windows originating from the same capture session from appearing in different splits. After the session-level split, LSTM sequences were generated using a sliding window of length 50 and step 5. The resulting partitions were used consistently for all evaluated models. Model performance was evaluated using Accuracy, Precision, Recall, Specificity, F1-score, Matthews Correlation Coefficient (MCC), and ROC-AUC, enabling comparison of neural architectures within the controlled experimental environment.

4 Proposed Architecture

This section describes the architecture implemented by the *local-lab* project [16]. The implementation is publicly available and version-controlled as part of the *local-lab* artifact. The architecture integrates four functional components: automated infrastructure deployment, controlled SlowDoS attack simulation, telemetry collection and dataset generation, and machine learning-based traffic analysis. By defining these components as version-controlled Infrastructure as Code artifacts, the environment can be reconstructed in a consistent manner for reproducible cybersecurity experimentation.

4.1 Local-Lab as a Reproducible Cloud-Native Security Architecture

Our *local-lab* defines the experimental infrastructure through an Infrastructure as Code workflow. Virtual machines, network topology, service deployment, and monitoring components are specified as version-controlled configuration artifacts and instantiated using OpenTofu [8] with the `libvirt` provider [18].

This design reduces configuration drift, supports consistent re-instantiation, and allows the infrastructure itself to be treated as a reusable scientific artifact. The deployed environment consists of three isolated nodes: a target node hosting the application stack, a monitoring node collecting telemetry, and an attacker node generating controlled malicious traffic. Figure 1 summarizes the architecture.

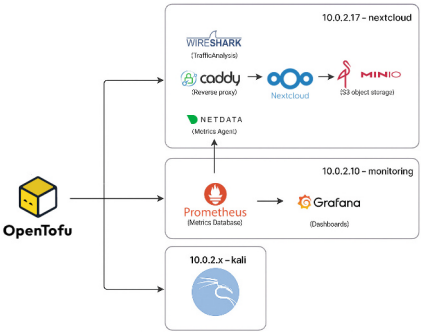

Fig. 1. Reproducible cloud-native architecture deployed using *local-lab* and OpenTofu.

4.2 Architectural Components and Node Roles

The architecture is composed of three nodes deployed within an isolated NAT-based network.

Target Node. The target node hosts Nextcloud behind a Caddy reverse proxy, with MinIO providing S3-compatible object storage. Packet-level traffic is captured using Wireshark or `tcpdump` [13], while Netdata collects system-level metrics such as CPU usage, memory consumption, open connections, and TCP socket states.

Monitoring Node. The monitoring node hosts Prometheus and Grafana. Prometheus scrapes metrics exposed by Netdata, while Grafana provides dashboards for observing system behavior during benign and malicious scenarios.

Attacker Node. The attacker node runs Kali Linux [15] and generates controlled malicious traffic against the target. Separating attacker, target, and monitoring roles enables realistic interaction patterns while preserving isolation from external networks.

4.3 Controlled Simulation of SlowDoS Attacks

Once the experimental platform is instantiated, controlled attack scenarios can be executed within the isolated network. Two representative SlowDoS techniques were evaluated:

- *Slowloris*, which maintains multiple incomplete HTTP header requests to exhaust server connection pools.
- *Slow POST*, generated using slowhttptest [22], which sends HTTP request bodies at extremely low transmission rates to keep server connections open for extended periods.

The attack execution process follows a methodology inspired by PTES [41], ensuring structured and systematic generation of malicious traffic. These attacks exploit protocol-level behavior rather than high bandwidth consumption, making them difficult to detect using traditional traffic-volume monitoring techniques. Within the proposed architecture, the attacker node generates malicious traffic toward the target node while the monitoring infrastructure records system behavior and network activity.

4.4 Telemetry Collection and Dataset Generation

A central goal of the proposed architecture is to enable systematic generation of cybersecurity datasets within a controlled and reproducible environment. Telemetry is collected through two complementary mechanisms. First, system and service metrics are captured using Netdata on the target node. Metrics such as TCP socket states, open connections, CPU utilization, and memory usage are periodically scraped by Prometheus and visualized through Grafana dashboards. Second, network traffic is captured at packet level using Wireshark or `tcpdump` under three operational conditions: normal application usage, Slowloris attack, and Slow POST attack. The resulting PCAP traces are processed using NTLFlowLyzer [23], which extracts statistical flow-level features from captured traffic. The extracted flows are labeled according to the corresponding operational scenario and exported as CSV datasets. The final dataset, SLOWDOS-2025, is obtained by merging balanced instances of benign and malicious traffic. Preprocessing includes duplicate removal, missing-value handling, feature engineering related to TCP handshake completion, and feature normalization using `StandardScaler`. By generating the dataset within a formally specified infrastructure, the resulting data remain tightly coupled with the experimental environment that produced them.

4.5 AI-Based Traffic Analysis

To evaluate intelligent detection within the proposed architecture, multiple neural network models were trained and evaluated on the generated dataset. An autoencoder was used to learn compressed latent representations of the traffic features, with embedding sizes of 128, 64, and 16 dimensions. These embeddings

were then used as inputs for supervised MLP classifiers. Finally, an LSTM network was introduced to capture temporal dependencies in network traffic flows. Unlike feed-forward models that operate on independent feature vectors, the LSTM processes sequences of flows using sliding windows, allowing the model to identify temporal patterns characteristic of slow denial-of-service attacks. The models were evaluated under different feature configurations, including the full feature set, a reduced feature subset, and autoencoder-generated embeddings. This setup enables analysis of the relationship between feature representation and detection performance.

4.6 Integrated Experimental Workflow

The overall experimental workflow enabled by the proposed architecture can be summarized as follows:

1. automated deployment of the cloud-native architecture using *local-lab*;
2. controlled execution of SlowDoS attacks within the isolated environment;
3. real-time monitoring and packet capture across infrastructure layers;
4. flow-based feature extraction and dataset generation;
5. data preprocessing and feature transformation;
6. training and evaluation of machine learning models.

By integrating reproducible infrastructure deployment, attack simulation, telemetry collection, and AI-based analysis, the proposed architecture provides a structured experimental framework for studying slow denial-of-service attacks in cloud-native environments.

5 Discussion and Limitations

5.1 Interpretation of the Experimental Results

Dataset Construction and Attack Effectiveness. The final dataset was obtained from PCAP captures collected under three operating conditions: benign traffic generated by a Nextcloud instance [9], and two SlowDoS attack scenarios produced using Slowloris [31] and slowhttptest [22]. Network traces were converted into tabular flow-based features through NTLFlowLyzer [23], resulting in a balanced binary dataset composed of Normal and SlowDoS traffic.

An important experimental observation concerns the different operational impact of the two slow-attack generators within the tested cloud-native stack. Slowloris connections were largely mitigated by the reverse proxy behavior, as Caddy [1] actively closes idle connections. In contrast, Slow POST attacks generated through slowhttptest were able to maintain partial requests and eventually produced service degradation. These observations show how the proposed architecture enables controlled observation of SlowDoS behavior across multiple infrastructure layers, including reverse proxy handling, application response, and network traffic characteristics. In this sense, the experimental setup demonstrates that a reproducible cloud-native security architecture can support systematic observation of application-layer SlowDoS dynamics and their operational impact, providing empirical evidence related to **RQ2**.

Role of Temporal Modeling with the Full Feature Set. When the full preprocessed feature set is used, the LSTM model achieves the best overall performance, with Accuracy 0.985, Precision 1.000, Recall 0.971, F1 0.985, and MCC 0.971, producing virtually no false positives while maintaining high sensitivity.

This result is consistent with the intrinsic characteristics of SlowDoS attacks. Rather than generating high traffic volumes, slow attacks exploit timing and persistence effects, such as incomplete requests or long-lived connections. Sequence-aware models are therefore able to capture dependencies between consecutive observations and recognize temporal attack patterns that static classifiers may partially overlook [35]. At the same time, the MLP configurations remain competitive, with MLP2 reaching AUC 0.992 and F1 0.966 on the full feature set. This indicates that aggregated flow statistics already encode a large portion of the discriminative signal extracted from network traffic [23]. The results therefore confirm that the dataset generated within the reproducible experimental environment enables systematic evaluation of AI-based detection models, supporting **RQ3**.

Impact of Feature Reduction. A consistent performance degradation is observed across all models when the FeatureMix dataset is used. For example, the LSTM accuracy decreases to 0.809 and the MCC drops to 0.618. This result suggests that the reduced feature subset discards information needed to represent the slow-attack dynamics within the evaluated cloud-native stack. SlowDoS attacks typically influence multiple correlated aspects of the communication process, including request completion patterns, connection duration, throughput behavior, and timing characteristics. When the representation becomes too sparse, classifiers lose complementary signals that jointly describe these behaviors.

Autoencoder Embeddings and Dimensionality Trade-Offs. The autoencoder-based dimensionality reduction provides insight into the amount of information required for reliable detection [36]. The results show a trade-off between compression and discriminative power. Latent representations of size 128 maintain high predictive performance, with the LSTM achieving an accuracy of 0.963 and an MCC of 0.927. A latent size of 64 offers an effective balance between compactness and accuracy, with MLP2 performing close to the best full-feature results. When the latent space is reduced to 16 dimensions, however, the loss of information becomes evident, indicating that aggressive compression removes fine-grained traffic characteristics required to avoid false positives and capture subtle attack behaviors.

Implications for Reproducible Cybersecurity Experimentation. Beyond the classification results, the experimental workflow demonstrates how cloud-native security architectures can be formally specified and instantiated as reproducible experimental environments.

The experimental infrastructure is defined through an Infrastructure as Code workflow based on OpenTofu and the `libvirt` provider [8,18]. The *local-lab* project [16] enables automated deployment of the same cloud-native stack used

throughout the experiments, allowing the infrastructure to be reconstructed in a consistent and version-controlled manner. Within this environment, monitoring and observation mechanisms operate across multiple layers. System and service metrics are collected through Netdata and exported to Prometheus and Grafana for time-series analysis [3,11,12], while network traffic is captured at packet level using Wireshark or `tcpdump` [13]. The combination of Infrastructure as Code provisioning, controlled traffic generation, and repeatable dataset extraction demonstrates that cloud-native security architectures can be instantiated as version-controlled experimental artifacts. This finding provides practical evidence related to **RQ1**.

Overall, the findings support the three research questions. The IaC-based deployment provides evidence for **RQ1**, the multi-layer observation of SlowDoS behavior supports **RQ2**, and the generated dataset enables the controlled evaluation of neural detection models required by **RQ3**.

5.2 Limitations and Threats to Validity

Operating System Scope. At present, the *local-lab* infrastructure [16] has been validated only on Linux-based hosts and guests. Extending the architecture to Windows or mixed operating-system environments would require additional base images, OS-specific configuration layers, and dedicated validation.

Scope and Generalizability. The evaluation focuses on a specific cloud-native stack composed of Nextcloud [9], Caddy [1], and MinIO [2]. Alternative reverse proxies, timeout policies, application stacks, or storage backends may affect cross-layer behavior, feature distributions, and model performance.

Attack Diversity. The experiments consider Slowloris [31] and slowhttptest-based Slow POST attacks [22]. Although these tools capture representative SlowDoS behaviors, additional application-layer and network-layer attacks should be considered in future work.

Dataset Realism and Workload Diversity. Benign traffic was generated through typical Nextcloud interactions, such as file uploads, downloads, and account operations. This workload is sufficient for controlled experimentation, but production environments exhibit greater variability in user behavior and traffic intensity.

Feature Extraction and Observability Limits. The detection pipeline relies on flow-based features extracted from PCAP traces using NTLFlowLyzer [23]. This supports network-level detection, but provides limited visibility into host-level and application-level signals such as process activity, application logs, or internal service states.

Statistical Robustness. The reported results correspond to a fixed experimental configuration and evaluation campaign, as shown in Table 2. Future work should exploit the repeatability of the platform to quantify run-to-run variability, perform additional repetitions, and report confidence intervals.

Table 2. Model performance for each feature configuration.

	ALL			FeatureMix			AE128			AE64			AE16		
Metric	MLP1	MLP2	LSTM	MLP1	MLP2	LSTM	MLP1	MLP2	LSTM	MLP1	MLP2	LSTM	MLP1	MLP2	LSTM
Accuracy	0.964	**0.965**	**0.985**	0.836	0.828	0.809	0.951	0.951	0.963	0.964	**0.965**	0.941	0.964	0.873	0.926
Precision	0.944	0.944	**1.000**	0.949	0.915	0.795	0.917	0.921	**0.985**	0.944	0.947	0.984	0.942	0.804	0.954
Recall	0.985	**0.988**	0.971	0.709	0.724	0.841	**0.991**	0.985	0.942	0.985	0.985	0.899	**0.988**	**0.988**	0.899
Specificity	0.942	0.942	**1.000**	0.962	0.724	0.776	0.910	0.915	**0.985**	0.942	0.945	**0.985**	0.938	0.758	0.955
F1-Score	0.964	**0.966**	**0.985**	0.812	0.808	0.817	0.953	0.952	0.963	0.964	**0.966**	0.939	0.965	0.887	0.925
AUC	0.990	**0.992**	0.987	0.936	0.911	0.894	**0.992**	**0.992**	0.984	0.989	**0.991**	0.982	0.987	0.990	0.957
MCC	0.928	**0.931**	**0.971**	0.722	0.6715	0.618	0.904	0.903	0.927	0.928	**0.931**	0.886	0.928	0.7672	0.855

Sequence Construction and Potential Data Leakage. The LSTM model uses overlapping temporal windows, which may introduce data leakage if windows originating from the same capture session are randomly distributed across training and test sets. To mitigate this risk, the split was performed at the PCAP capture-session level before window generation. All flows extracted from the same capture were assigned to a single partition, and sliding windows were then constructed independently within each partition. This protocol prevents overlapping or near-duplicate windows from the same capture session from being shared across different evaluation splits.

6 Conclusion and Future Work

This work examined how cloud-native security infrastructures can be specified, reconstructed, and reused to support controlled cybersecurity experimentation and AI-based detection model evaluation. The proposed *local-lab* architecture integrates Infrastructure as Code deployment, controlled SlowDoS attack generation, multi-layer telemetry collection, packet capture, dataset construction, and machine learning analysis within a unified experimental pipeline. The architecture was implemented using OpenTofu and the `libvirt` virtualization backend to instantiate a cloud-native stack composed of Nextcloud, Caddy, and MinIO, with monitoring provided by Netdata, Prometheus, and Grafana. Packet captures collected under benign, Slowloris, and Slow POST scenarios were processed through NTLFlowLyzer to generate the labeled SLOWDOS-2025 dataset. The results show that the proposed environment enables controlled observation of SlowDoS behavior across multiple infrastructure layers and provides a structured setting for evaluating AI-based detection models. In particular, the LSTM model achieved the best performance on the full feature set, confirming the relevance of temporal modeling for low-rate attacks characterized by persistent connections and delayed request completion. More broadly, the study demonstrates that cloud-native security architectures can be formally described and instantiated as version-controlled artifacts through Infrastructure as Code workflows. This design supports transparent, repeatable, and comparable cybersecurity experimentation, addressing the reproducibility limitations of studies that release only model code or datasets. Future work will extend the architecture to public cloud

platforms, multi-platform deployments, additional attack classes, and container-based or Kubernetes-based environments. These extensions will allow the proposed methodology to be evaluated under larger, more heterogeneous, and more realistic cloud-native scenarios.

References

1. Caddy Server: Caddy Reverse Proxy. https://github.com/caddyserver/caddy
2. MinIO: MinIO Object Storage. https://github.com/minio/minio
3. Netdata: Netdata Monitoring. https://github.com/netdata/netdata
4. Ylönen, T.: Secure Shell Protocol. RFC 4250 (1995). https://www.rfc-editor.org/rfc/rfc4250
5. Tenable Inc.: Nessus Vulnerability Scanner (2023). https://www.tenable.com/downloads/nessus
6. CVE Details (2025). https://www.cvedetails.com/vendor/15913/
7. OpenCVE (2025). https://app.opencve.io/cve/?vendor=nextcloud
8. Linux Foundation: OpenTofu (2025). https://opentofu.org/
9. Nextcloud GmbH.: Nextcloud (2025). https://nextcloud.com
10. Ambrosino, C.: Nextcloud with S3 Object Storage (2025). https://github.com/carmine-ambrosino/nextcloud-s3
11. Prometheus Monitoring System (2025). https://prometheus.io
12. Grafana Labs: Grafana (2025). https://grafana.com
13. Wireshark Team: Wireshark Network Analyzer (2025). https://www.wireshark.org
14. Oracle Corporation: VirtualBox (2025). https://www.virtualbox.org
15. Kali Linux (2025). https://www.kali.org
16. Ambrosino, C.: local-lab: Cloud Lab Locally (2025). https://github.com/carmine-ambrosino/local-lab
17. Lubuntu Project: Lubuntu (2025). https://lubuntu.me
18. dmacvicar: Terraform Provider for Libvirt (2025). https://github.com/dmacvicar/terraform-provider-libvirt
19. Red Hat, et al.: Fedora Workstation (2025). https://fedoraproject.org
20. Sarti, M.: How to run virtual machines with virt-manager. Fedora Mag. (2025). https://fedoramagazine.org/full-virtualization-system-on-fedora-workstation-30/
21. Ettari, S.: System Monitoring Stack (2025). https://github.com/simona2606/system-monitoring-stack
22. Shekyan, S.: SlowHTTPTest (2025). https://github.com/shekyan/slowhttptest
23. Shafi, M.M., Lashkari, A.H., Roudsari, A.H.: NTLFlowLyzer: towards generating an intrusion detection dataset and intruders behavior profiling through network and transport layers traffic analysis and pattern extraction. Comput. Secur. **148** (2025). https://doi.org/10.1016/j.cose.2024.104160
24. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery in databases. AI Mag. **17**(3), 37 (1996)
25. Munaiah, N., Meneely, A., Eriksson, B., Williams, L.: A survey of cyber ranges and testbeds. Comput. Secur. **88**, 101636 (2019)
26. Yamin, M.M., Katt, B., Gkioulos, V.: Cyber range: the next generation cyber security training platform. Comput. Secur. **97**, 101961 (2020)
27. Nakata, T., Otsuka, A.: Cyber range architecture using container-based virtualization. In: Proceedings of the IEEE International Conference on Cyber Conflict (2018)

28. Tavakoli, M., Smith, P.: Automatic generation of cyber ranges for security experimentation. J. Cybersecur. **7**(1) (2021)
29. Kyriakopoulos, K., Theocharidou, M., Gritzalis, S.: Cyber range platforms and testbeds for cybersecurity research: a survey. IEEE Access **10**, 109512–109534 (2022)
30. Guirguis, M., Bestavros, A., Matta, I., Zhang, Y.: Low-rate TCP-targeted denial of service attacks and counter strategies. In: Proceedings of the IEEE International Conference on Network Protocols (2005)
31. Russell, R.: Slowloris HTTP Denial of Service Attack (2010). https://ha.ckers.org/slowloris/
32. Sivaguru, R., Nithyanand, R.: Detection and mitigation of low-rate denial-of-service attacks: a survey. IEEE Access **10**, 89764–89790 (2022)
33. Shekyan, S.: Experimental evaluation of slow HTTP denial-of-service attacks. Int. J. Inf. Secur. **17**(5), 497–510 (2018)
34. Javaid, A., Niyaz, Q., Sun, W., Alam, M.: A deep learning approach for network intrusion detection system. In: Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (2016)
35. Yin, C., Zhu, Y., Fei, J., He, X.: A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access **5**, 21954–21961 (2017)
36. Sakurada, M., Yairi, T.: Anomaly detection using autoencoders with nonlinear dimensionality reduction. In: Proceedings of the MLSDA Workshop at ACM SIGKDD (2014)
37. Aladaileh, M., Anbar, M., Hasbullah, H.: Deep learning based detection of slow HTTP denial of service attacks. J. Netw. Comput. Appl. **150**, 102485 (2020)
38. Yu, S., Zhou, W., Doss, R., Jia, W.: Traceback of DDoS attacks using entropy variations. IEEE Trans. Parallel Distrib. Syst. **22**(3), 412–425 (2011)
39. Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.: A detailed analysis of the KDD CUP 99 data set. In: IEEE Symposium on Computational Intelligence for Security and Defense Applications (2009)
40. Sharafaldin, I., Lashkari, A., Ghorbani, A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: Proceedings of the International Conference on Information Systems Security and Privacy (ICISSP) (2018)
41. Cloudflare, Inc.: Penetration Testing Execution Standard. http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines
42. Ficco, M., Rak, M.: Stealthy denial of service strategy in cloud computing. IEEE Trans. Cloud Comput. **3**(1), 80–94 (2015). https://doi.org/10.1109/TCC.2014.2325045