Detection Attack DOS Using Reinforcement Learning

Abstract

In light of the increasing prevalence of Denial-of-Service (DoS) attacks, robust detection mechanisms have become essential to ensure the availability and reliability of computer networks. This thesis proposes a novel intrusion detection approach that leverages Reinforcement Learning (RL) to identify and mitigate DoS attacks. The RL agent learns optimal detection policies by interacting with a simulated network environment, continuously improving its performance through reward-based feedback. Key network traffic features are extracted and used as state inputs for the agent, enabling it to distinguish between benign and malicious activity. The model is evaluated in benchmark datasets such as CSE-CIC-IDS2018, demonstrating high detection accuracy, low false positive rates, and adaptability to evolving attack patterns. Compared to traditional machine learning classifiers, the RL-based system shows improved response and decision-making capabilities under dynamic network conditions. Future research will explore the integration of this system into real-time network infrastructures and the enhancement of its scalability for broader threat detection.

Keywords: Intrusion Detection Systems (IDS), Denial-of-Service (DoS), Reinforcement learning (RL), CSE-CIC-IDS2018, Cybersecurity, Network Traffic Analysis.

Contents

Chapter 1

Intrusion Detection System (IDS)

1.1 Security Fundamentals

Introduction

Security fundamentals refer to the essential principles, concepts, and practices that form the foundation of information security. These fundamentals encompass a wide range of technical and organizational measures aimed at protecting sensitive information and systems from unauthorized access, theft, damage, or other forms of compromise.

Key security fundamentals include Confidentiality, Integrity, Availability, Authentication, Authorization, Encryption, Risk Management, Incident Response, and Disaster Recovery. Together, these principles establish the basis of a comprehensive information security program, enabling organizations to effectively safeguard their critical information assets and maintain the trust of stakeholders.[1]

1.1.1 Core Security Concepts

Network security encompasses strategies and technologies to protect systems from cyber threats, particularly Denial of Service (DoS) attacks that aim to disrupt service availability. Several core principles serve as the foundation for secure systems.

CIA Triad

Confidentiality Confidentiality ensures that sensitive information is only accessible to authorized users. Techniques such as encryption, user authentication, and access control policies prevent unauthorized data access. While DoS attacks do not typically aim to breach confidentiality directly, successful exploitation may lead to indirect confidentiality violations if attackers cause service misconfigurations or force failovers to insecure states.

Integrity Integrity guarantees that data is accurate and unaltered. It is maintained through cryptographic hash functions, checksums, and digital signatures that validate whether data has been tampered with. During a DoS attack, integrity may be compromised by interrupting legitimate updates or corrupting processes due to system overload.

Availability Availability ensures that services and systems remain accessible to legitimate users at all times. This principle is the primary target of DoS attacks, which flood networks or services with excessive requests, causing slowdowns or complete denial of access. Maintaining availability involves redundancy, load balancing, and proactive mitigation strategies like rate limiting and firewalls.

Figure 1.1: Principles of information security

Source: [2]

Extended Security Principles

Non-repudiation Non-repudiation guarantees that an entity cannot deny having performed a particular action, such as sending a message or initiating a transaction. This is enforced through digital signatures and secure logging mechanisms. In the context of DoS attacks, non-repudiation helps trace attack origins and supports legal accountability.

Authenticity Authenticity confirms that data, communications, or users are genuine and not forged. Authentication protocols, digital certificates, and cryptographic techniques are used to ensure that data comes from trusted sources. This is crucial for filtering legitimate traffic from spoofed attack traffic in DoS scenarios.

Accountability Accountability ensures that all actions within a system can be traced to responsible users or processes. It involves logging, auditing, and monitoring to track behavior. Accountability is vital for forensic analysis after a DoS attack and for strengthening defenses against future intrusions.

1.2 Denial-of-Service (DoS) Attacks

Introduction Denial-of-Service (DoS) attacks aim to make a system or network resource unavailable to its intended users by overwhelming it with excessive traffic or exploiting protocol-level vulnerabilities. These attacks can disrupt services, degrade performance, or completely shut down access. The most common types include:

1.2.1 Volumetric Attacks

Volumetric attacks aim to saturate a target's bandwidth by generating an overwhelming amount of traffic. This often involves amplification techniques or high-rate packet floods. Common examples include UDP floods and DNS amplification attacks, which leverage misconfigured servers to multiply traffic directed at the victim [3].

In a UDP flood, attackers send large numbers of spoofed UDP packets to random or specific ports, consuming bandwidth and processing power. DNS amplification uses small queries to open resolvers with the victim's IP address, causing them to return large responses to the victim.

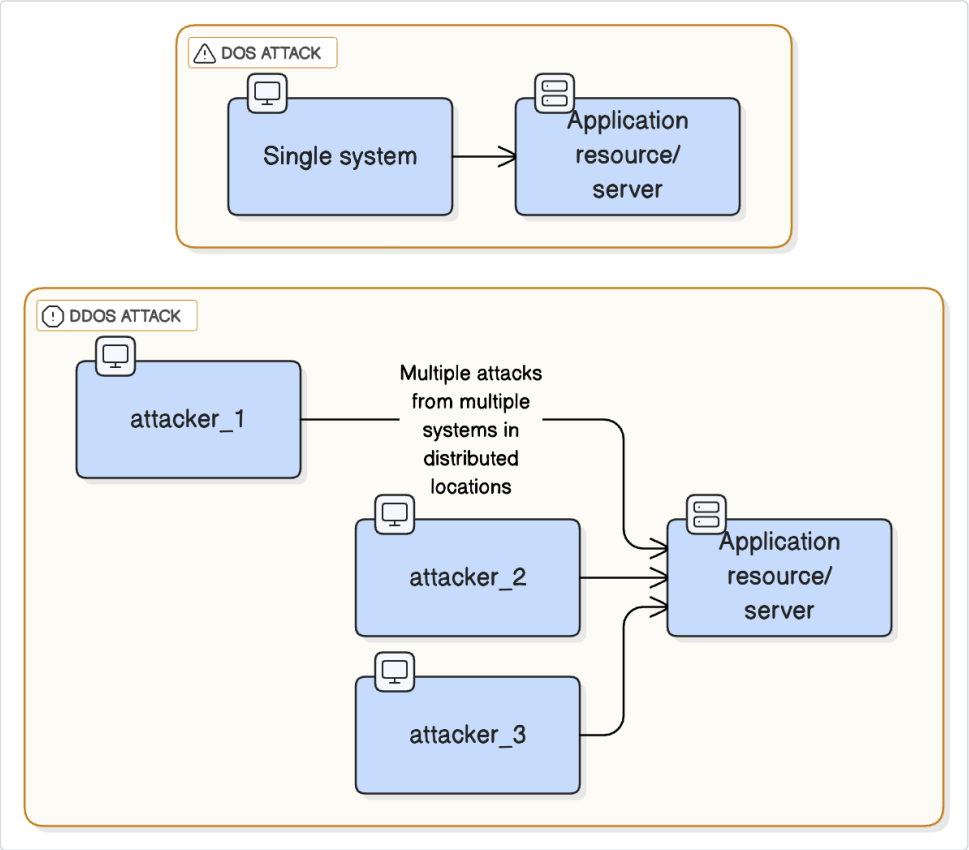

Figure 1.2: DoS and DDoS Attack Traffic Comparison

A notable example is the Mirai botnet, which infected hundreds of thousands of insecure IoT devices to coordinate massive traffic streams toward its targets [3]. The goal of volumetric attacks is to clog network links (Gbps or Tbps scale), preventing legitimate access.

1.2.2 Protocol Attacks

Protocol attacks exploit weaknesses in Layer 3/4 protocols (network or transport layer) to exhaust server or network resources. Unlike volumetric attacks, they do not require high bandwidth but instead consume stateful resources like connection tables or CPU cycles [3].

A classic example is the TCP SYN flood, where attackers send numerous SYN packets without completing the handshake, filling the server's connection queue [4]. ICMP floods and Ping of Death attacks exploit the Internet Control Message Protocol to crash or freeze systems by sending malformed or excessive traffic.

Other examples include fragmentation attacks (e.g., Teardrop), which send overlapping IP fragments to crash reassembly logic, and ACK/FIN floods that exhaust firewall state tables.

1.2.3 Application Layer Attacks

Application-layer (Layer 7) attacks generate traffic that appears legitimate at the application level but consumes excessive server resources such as CPU, memory, or threads [3]. These attacks are difficult to detect because they mimic real user behavior.

HTTP GET/POST floods can overwhelm web servers by triggering costly operations. A well-known example is Slowloris, which sends partial HTTP headers slowly to hold open many connections and exhaust the web server's pool [5]. Other examples include RUDY (R-U-Dead-Yet) attacks and HTTP floods targeting dynamic or database-backed content.

1.2.4 Distributed Denial-of-Service (DDoS)

DDoS attacks use multiple compromised machines (botnets) to launch coordinated attacks, making them harder to block and vastly more powerful than single-source DoS attacks [3].

Botnets like Mirai leverage IoT devices to simultaneously launch volumetric or protocol-based attacks. Reflective amplification (e.g., using open DNS/NTP servers) further increases traffic impact. Mitigation requires upstream filtering, rate-limiting, and often third-party scrubbing services.

1.2.5 Network Traffic Analysis

Mitigating DoS/DDoS attacks relies on monitoring and anomaly detection. This involves establishing a baseline of normal network behavior and identifying deviations in volume, protocol use, or source IPs [3, 6].

Anomaly-based intrusion detection systems (IDS) can flag unusual spikes, connection patterns, or TCP flag anomalies. Signature-based systems match known attack patterns. Flow analysis tools (e.g., NetFlow, sFlow) help detect irregular byte/packet rates or unusual source-destination pairs.

Advanced methods include machine learning to classify deviations. Once detected, defenses such as rate limiting, traffic shaping, or redirection to mitigation services are employed to maintain availability.

1.3 Intrusion Detection Systems (IDS)

An **Intrusion Detection System (IDS)** is a cybersecurity solution designed to monitor network or system activities for malicious actions or policy violations. Upon detecting such activities, the IDS typically alerts system administrators or integrates with centralized security tools like Security Information and Event Management (SIEM) systems to facilitate a coordinated response [7].

1.3.1 IDS Architecture

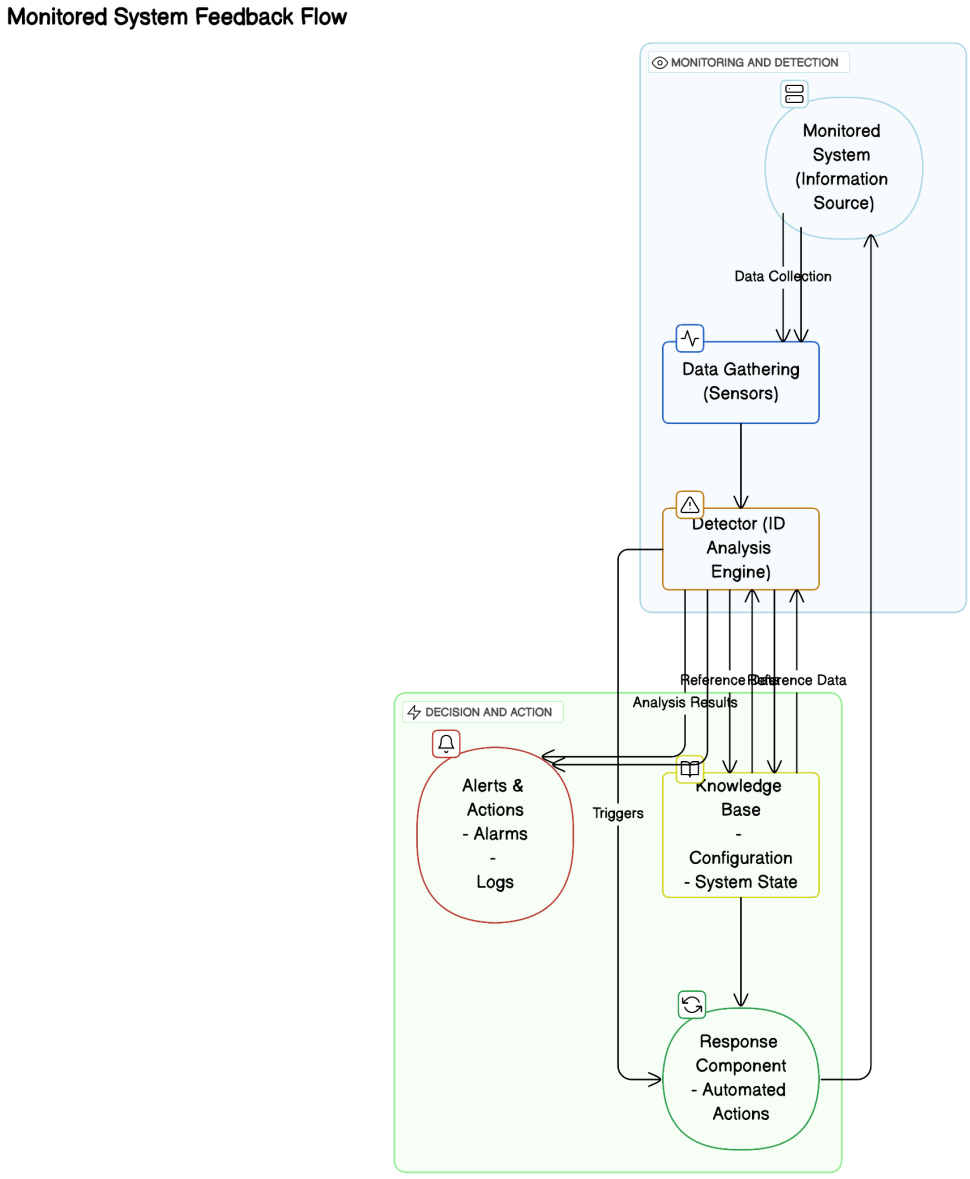

Figure 1.3: Network-based Intrusion Detection System (NIDS) Architecture

Modern IDS platforms implement modular architectures with several key components:

- **Sensors:** are the front line of an IDS, deployed at strategic network chokepoints—such as mirror (SPAN) ports, network taps, or virtual interfaces in cloud environments—to collect raw traffic data. They capture full packet streams, flow records, or application logs and often include built-in filters or sampling mechanisms to reduce noise in high-volume settings. By pre-processing and forwarding only relevant events, sensors ensure the IDS receives a representative yet manageable dataset, enabling visibility into lateral movement, data exfiltration, and denial-of-service attempts without overwhelming the analysis layer.

- **Analysis Engines:** Analysis Engines form the IDS's "brain," taking the sensor-collected data and

applying a mixture of rule-based and intelligent techniques to detect threats. Traditional pattern-matching engines compare payloads against known attack signatures, while statistical or machine-learning analyzers establish behavioral baselines and flag deviations. Protocol analyzers dive deep into application-level conversations—HTTP, DNS, SMB, etc.—to spot malformed requests or anomalous sequences. By correlating events across multiple sensors and data sources, the engine can piece together multi-stage attacks (e.g., a scan followed by an exploit) and prioritize alerts based on risk.

- **Knowledge Base:** The Knowledge Base underpins all detection logic by maintaining up-to-date signatures, behavioral profiles, and historical datasets. Signature databases are refreshed with the latest threat intelligence feeds, while anomaly detectors continuously refine their statistical models using both live traffic and feedback on past alerts. In more advanced platforms, machine-learning feedback loops incorporate security analyst verdicts—true positive, false positive—to retrain the system, improving accuracy over time. This shared repository ensures that the IDS can recognize both known exploits and subtle shifts in normal network behavior.

- **Response Systems:** Response Systems close the loop between detection and defense. Once the analysis engine assigns a confidence score to an event, the response component generates alerts—pushing them into dashboards, SIEMs, or ticketing systems—and, where policies allow, triggers automated countermeasures. High-confidence detections might invoke firewall rule updates, IP blacklists, or host quarantines, while lower-confidence events are routed to alert managers for human review. Integrated visualization tools help security teams triage incidents rapidly, and orchestration connectors enable the IDS to participate in broader security workflows, ensuring a coordinated, efficient reaction—especially critical when facing large-scale DoS or DDoS onslaughts.

For DOS attack detection, these components must operate with high efficiency and scale to process enormous traffic volumes during attack scenarios.

1.3.2 IDS Types and Functionality

Intrusion Detection Systems serve as the primary monitoring and detection layer for network security threats, including DOS attacks:

- **Network-based IDS (NIDS):** A Network-based Intrusion Detection System (NIDS) is a monitoring solution placed at key junctions within a network—often on a mirrored switch port or network tap—where it passively captures and inspects all passing traffic. Unlike host-based agents, a NIDS doesn't rely on software installed on individual machines; instead, it reconstructs network sessions and analyzes packet headers and payloads in real time. It uses signature-based detection (comparing packets against a database of known attack patterns) and anomaly-based detection (profiling normal traffic volumes, protocols, and behavioral baselines) to spot suspicious or malicious activity—such as port scans, denial-of-service floods, SQL injection attempts, or data exfiltration. When the NIDS flags a potential intrusion, it generates an alert that can be logged centrally, forwarded to a Security Information and Event Management (SIEM) system, or used to trigger automated defenses (e.g., instructing a firewall to block the offending IP). Because it sees all traffic crossing its monitored segment, a properly tuned NIDS provides broad visibility into attacker reconnaissance and network-level exploits—though it can be blind to encrypted payloads unless integrated with SSL/TLS decryption, and it may generate false positives if thresholds or signatures aren't carefully calibrated. By complementing host-based sensors and other controls, a NIDS forms a crucial layer in a network's defense-in-depth strategy.

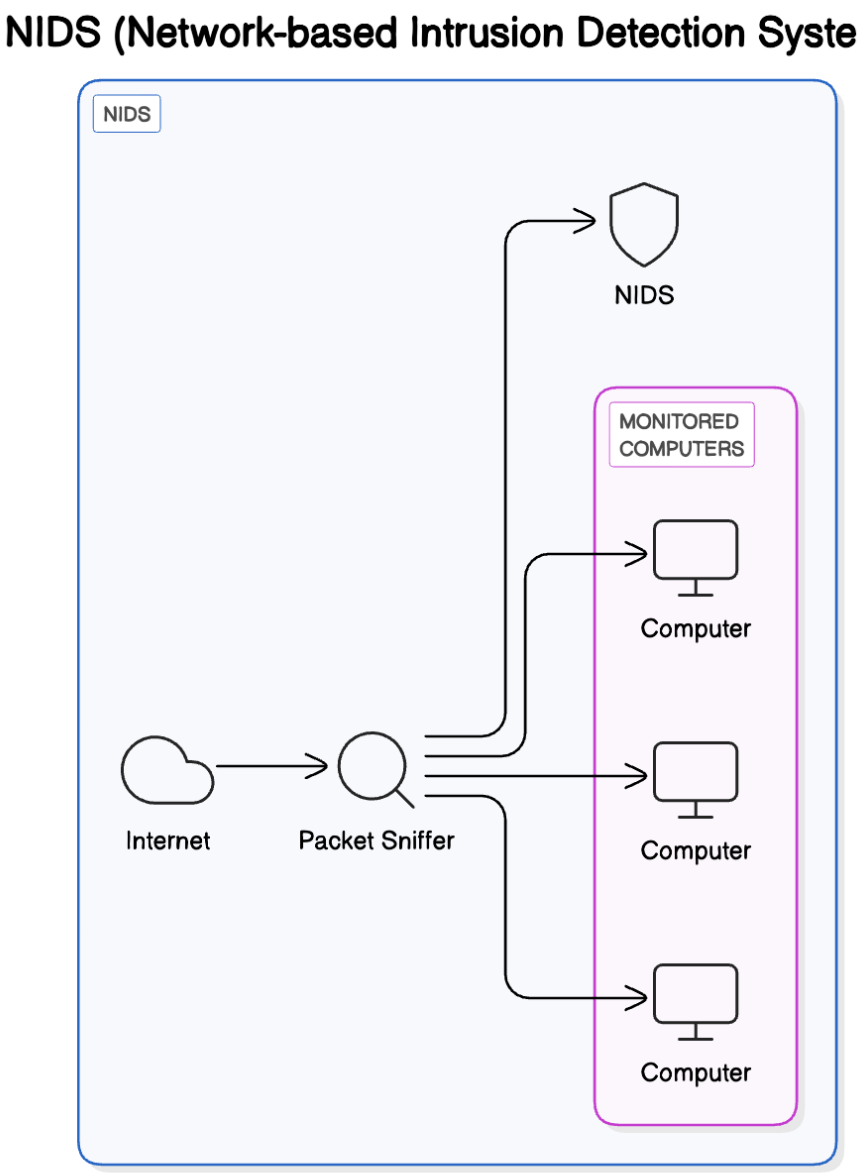

Figure: Network-based Intrusion Detection System (NIDS) Architecture

- **Host-based IDS (HIDS):** A Host-based Intrusion Detection System (HIDS) operates at the level of individual endpoints or servers, offering deep visibility into system-specific activities. Instead of analyzing network traffic, HIDS monitors internal events such as system calls, file integrity changes, registry modifications, user logins, and local log files. This allows it to detect unauthorized alterations, privilege escalations, malware activity, and localized impacts of denial-of-service (DoS) attacks that may not be evident from a network perspective. By focusing on what happens inside the host, HIDS provides rich contextual information—such as which process triggered a suspicious action or which user account was involved—which is crucial for forensic analysis and containment. However, its scope is inherently limited to the device it protects, offering little visibility into lateral movement or network-wide attack patterns. Common examples of HIDS tools include OSSEC, Tripwire, and Wazuh.

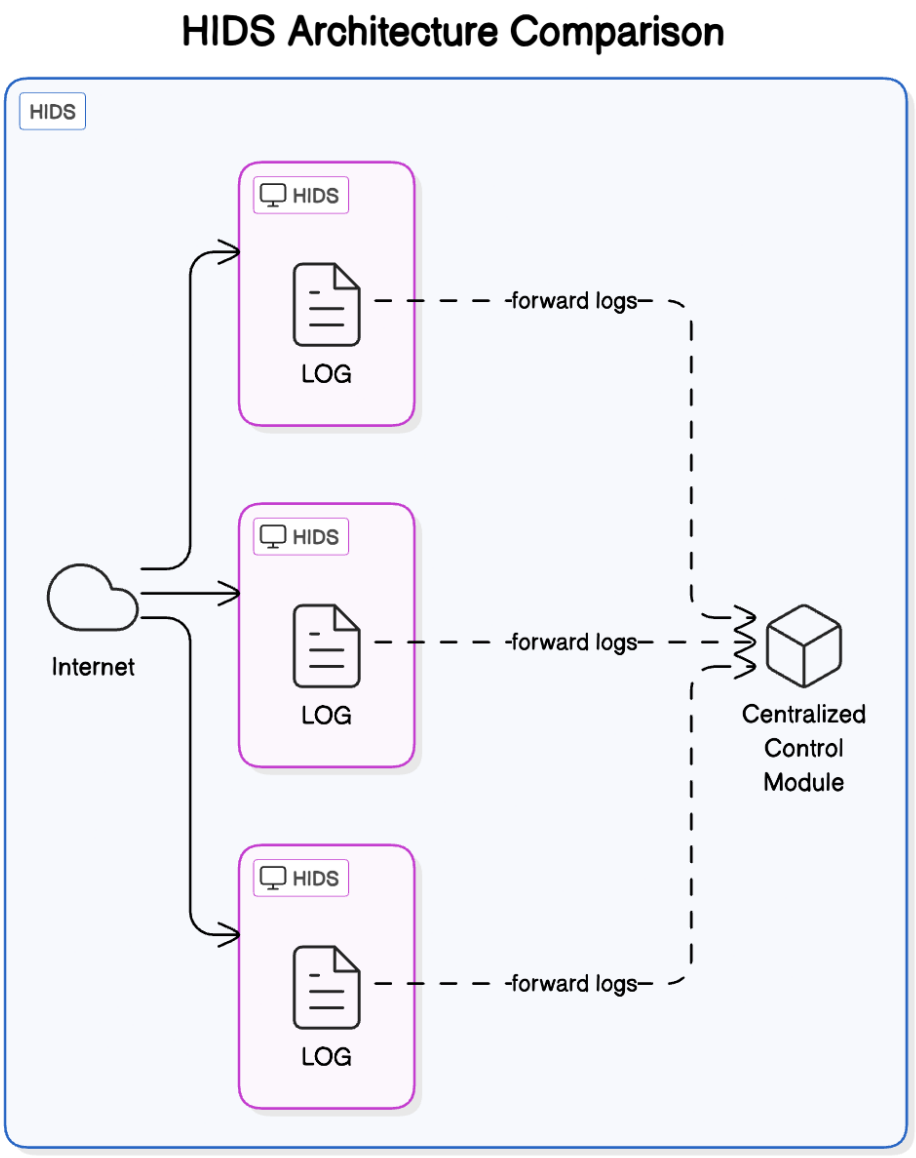

Figure: Network-based Intrusion Detection System (NIDS) Architecture

- **Detection Methodologies:**
  - **Signature-based:** Matches observed traffic against known attack patterns (signatures)
    - Highly effective for known attacks with clear signatures
    - Requires regular signature updates
    - Ineffective against zero-day or modified attacks
    - Examples: SYN flood patterns, known botnet command signatures
  - **Anomaly-based:** Establishes baselines of normal behavior and flags significant deviations
    - Can detect previously unknown attack vectors
    - Requires training period to establish accurate baselines
    - Typically generates more false positives than signature-based systems
    - Examples: Traffic volume spikes, unusual protocol distributions

- **Hybrid Systems:** Combine signature and anomaly detection approaches
  - Leverage strengths of both methodologies
  - Use signatures for known threats and anomaly detection for novel attacks
  - Implement weighted alert systems for confidence scoring
  - Reduce false positives through correlation of multiple detection methods

Limitations of Traditional IDS for DOS attack detection include:

- Difficulty processing high-volume traffic during attacks
- Challenges distinguishing flash crowds from attacks
- Limited adaptation to evolving attack techniques
- High false-positive rates with anomaly detection
- Resource consumption during high-traffic periods

These limitations necessitate AI-driven approaches like reinforcement learning that can adapt to evolving threat landscapes.

1.4 Classification in Security Contexts

1.4.1 Classification Paradigms

Classification forms the foundation of automated threat detection, organizing network traffic into categories for analysis and response:

Binary Classification: The simplest approach for distinguishing between normal and attack traffic. This method provides clear decision boundaries for basic filtering with low granularity but high processing efficiency. It is essential for initial traffic triage during high-volume attacks, offering straightforward pass/block decisions and benign/malicious categorization.

Multi-class Classification: A more sophisticated approach that distinguishes between multiple attack types, enabling targeted responses for specific threats. While requiring more complex models and training data, this method supports detailed attack attribution and can differentiate between various attack vectors such as SYN floods, HTTP floods, and DNS amplification attacks.

Hierarchical Classification: An organized approach that structures threats in a tree-like format, enabling progressive refinement of classifications. This method balances processing efficiency with detection detail and supports multi-stage detection pipelines. For example, traffic can be progressively classified as: Traffic $\rightarrow$ Attack $\rightarrow$ DOS $\rightarrow$ Protocol-based $\rightarrow$ SYN Flood.

Multi-label Classification: An advanced approach that assigns multiple categories to a single traffic flow, recognizing attacks with multiple characteristics. This method identifies complex attack campaigns and supports sophisticated response orchestration. For instance, traffic can be simultaneously classified as "volumetric," "distributed," and "amplification."

Classification outcomes drive security responses, determining whether traffic should be allowed, throttled, redirected, or blocked entirely.

What is Machine Learning?

Machine learning (ML) is a branch of artificial intelligence (AI) focused on enabling computers and machines to imitate the way that humans learn, to perform tasks autonomously, and to improve their performance and accuracy through experience and exposure to more data.

According to UC Berkeley, the learning system of a machine learning algorithm can be broken down into three main parts:

1. **A Decision Process:** In general, machine learning algorithms are used to make a prediction or classification. Based on some input data, which can be labeled or unlabeled, the algorithm produces an estimate about a pattern in the data.

2. **An Error Function:** An error function evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.

3. **A Model Optimization Process:** If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm repeats this iterative "evaluate and optimize" process, updating weights autonomously until a threshold of accuracy has been met.

Source: https://www.ibm.com/think/topics/machine-learning

1.4.2 Machine Learning Techniques

Modern security systems leverage diverse machine learning approaches for traffic classification:

Supervised Learning

Supervised learning involves training on labeled datasets where the "ground truth" classifications of attacks are already known. This approach allows models to learn from past data and make accurate predictions on new, unseen inputs. Common algorithms used in supervised learning include Random Forests, which are ensembles of decision trees that offer robust classification; Support Vector Machines (SVM), which are effective at identifying decision boundaries between different types of network traffic; Neural Networks, including multi-layer perceptrons and deep learning architectures capable of recognizing complex patterns; and Gradient Boosting methods like XGBoost, which are known for their high-performance classification capabilities. Supervised learning is especially useful in scenarios such as detecting known attacks using labeled training data, identifying specific types of attacks, analyzing the importance of different features, and deploying models in production environments where minimizing false positives is crucial.

Unsupervised Learning

Unsupervised learning focuses on uncovering patterns and structures within data that has not been labeled. This approach is especially useful when dealing with vast amounts of network traffic where manual labeling is impractical or impossible. Techniques commonly used in unsupervised learning include clustering methods such as k-means and DBSCAN, which group similar traffic patterns together based on statistical similarity. Anomaly detection methods like Isolation Forest and One-Class SVM are effective for identifying outliers that may indicate potential security threats. Dimensionality reduction techniques such as Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) are used to simplify complex traffic data, making it easier to analyze. Autoencoders, a type of neural network, learn representations of normal traffic behavior and can highlight anomalies when deviations occur. Unsupervised learning is particularly valuable for identifying novel attack patterns, establishing baselines for normal behavior, detecting zero-day attacks, and categorizing traffic without the need for prior labeling.

Semi-Supervised Learning

Semi-supervised learning bridges the gap between supervised and unsupervised approaches by combining a small amount of labeled data with a much larger pool of unlabeled data. This method reduces the need for extensive manual labeling while still achieving reasonable accuracy in classification. It is especially beneficial in cybersecurity contexts where labeled attack data may be scarce, but large amounts of raw traffic data are available. Techniques in this category include self-training classifiers, where the model iteratively labels new data based on its predictions, and label propagation methods, which spread label information across a data

graph based on similarity. Semi-supervised learning is particularly valuable in dynamic and evolving threat landscapes, where continuous adaptation to new types of attacks is required without exhaustive annotation efforts.

These approaches also form the foundation for reinforcement learning systems, which build upon the outcomes of classification tasks to develop and refine optimal security policies.

Chapter 2

Reinforcement Learning Applications: A Comprehensive Guide

2.1 Introduction to the Reinforcement Learning Approach

Reinforcement Learning (RL) is a branch of machine learning that focuses on how agents can learn to make decisions through trial and error to maximize cumulative rewards. Unlike supervised learning, where models learn from labeled data, RL involves learning optimal behaviors through interactions with an environment, receiving feedback in the form of rewards or penalties based on actions taken. This approach enables agents to discover strategies that yield the highest long-term benefits, making RL particularly effective for tasks involving sequential decision-making, such as robotics, game playing, and autonomous systems. Reinforcement Learning provides several key advantages over traditional DoS detection approaches:
Source: Adapted from GeeksforGeeks: What is Reinforcement Learning

2.2 Introduction to Deep Q-Learning

Deep Q-Learning is a powerful extension of the traditional Q-Learning algorithm that leverages deep neural networks to approximate the Q-value function. In standard Q-Learning, an agent maintains a Q-table that maps state-action pairs to expected future rewards. However, this becomes impractical for environments with large or continuous state spaces. Deep Q-Learning addresses this limitation by using a deep neural network, known as a Q-network, to estimate Q-values directly from raw input states.

In reinforcement learning, the agent interacts with the environment in discrete time steps. At each step t, the agent observes a state s_t, selects an action a_t, receives a reward r_t, and transitions to a new state s_{t+1}. The goal is to learn a policy π that maximizes the expected cumulative reward over time.

The Q-value function $Q(s, a)$ represents the expected return of taking action a in state s and following the policy thereafter. In Deep Q-Learning, the Q-network is trained to minimize the difference between the predicted Q-value and the target Q-value, which is computed using the Bellman equation:

$$Q(s\_t, a\_t) = r\_t + \gamma \max\_a' Q(s\_t + 1, a'; \theta^-) \tag{2.1}$$

Here, γ is the discount factor, θ are the parameters of the current Q-network, and θ^- are the parameters of a target network that is periodically updated to stabilize training.

To improve learning stability and efficiency, Deep Q-Learning introduces two key techniques:

- **Experience Replay:** Stores past experiences in a replay buffer and samples mini-batches randomly during training to break correlations between consecutive samples.
- **Target Network:** Uses a separate, periodically updated target network to compute the target Q-values, reducing oscillations and divergence.

Deep Q-Learning has been successfully applied to various complex decision-making tasks, such as playing Atari games directly from raw pixels, demonstrating its potential in handling high-dimensional input spaces.

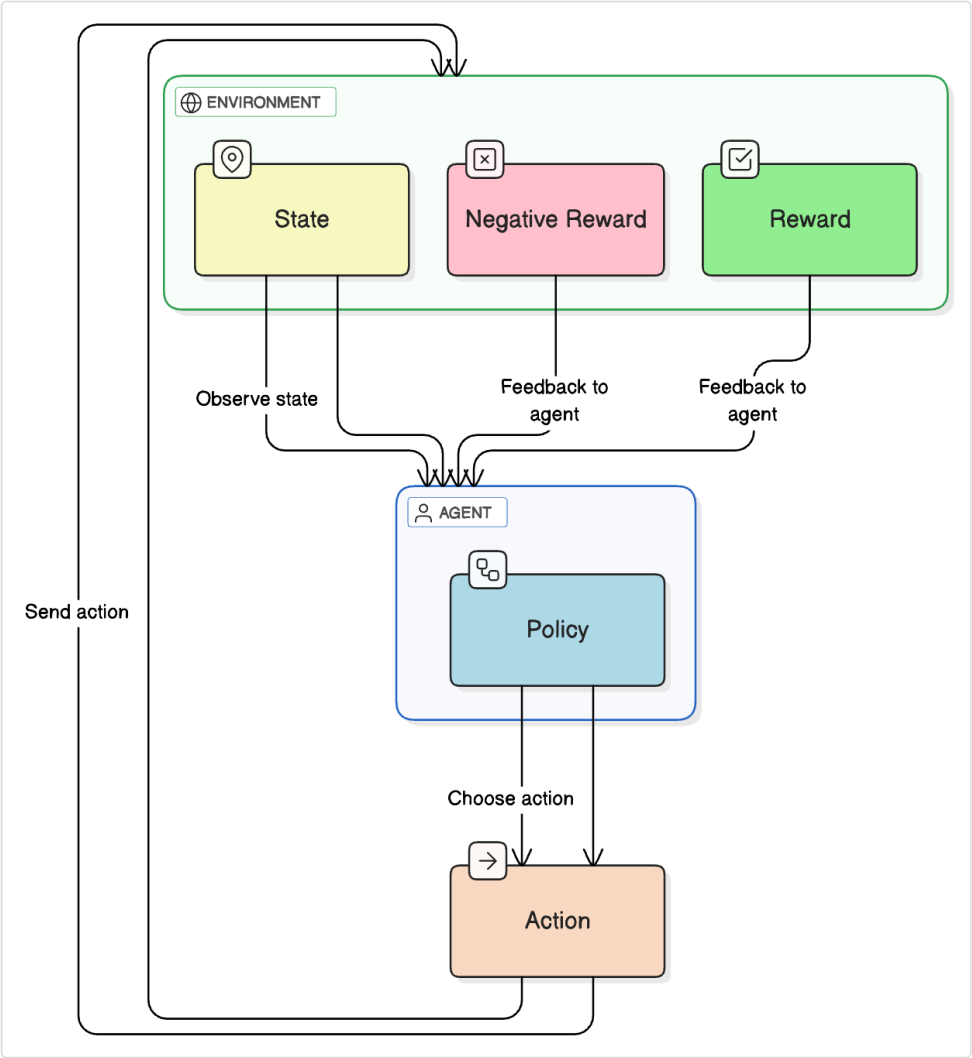

Figure: Basic Components of Reinforcement Learning

2.3 Environment Modeling for DoS Detection

The foundation of any RL application is a well-designed environment that accurately represents the problem domain. For DoS detection, this environment must model network traffic patterns and attack scenarios in a way that enables effective learning.

2.3.1 State Space Design

The state space in a reinforcement learning (RL) framework for Denial-of-Service (DoS) detection encodes the essential characteristics of network traffic that the agent uses to make decisions. A well-designed state space should reflect a comprehensive yet efficient representation of the network environment, enabling the RL agent to distinguish between normal and malicious activity. This design typically integrates four categories of features: traffic volume metrics, statistical distribution properties, temporal behavior indicators, and content-based descriptors. Each category offers a unique perspective on the nature of the traffic, contributing to a robust and informative state representation.

Traffic volume features quantify the scale and frequency of network activity. These include metrics such as packets per second, bytes per second, flow initiation rates, and the number of concurrent connections. These indicators are particularly useful for detecting volumetric DoS attacks, which are characterized by sudden spikes in traffic volume designed to overwhelm network resources. By monitoring these patterns, the RL agent can quickly recognize and react to abnormal surges in activity.

Statistical distribution features capture how traffic attributes are spread across various dimensions. Entropy values of source and destination IP addresses, protocol usage percentages, port distributions, and packet size variation are key indicators in this category. These features help identify anomalies that are not necessarily reflected in overall traffic volume but manifest as irregularities in distribution patterns, such as a high concentration of traffic targeting a specific port or originating from a narrow set of addresses.

Temporal pattern features provide insight into the evolution of traffic over time. By analyzing short-term trends, comparing current behavior to historical baselines, or evaluating periodicity, the agent can detect subtle or persistent attack patterns. Features such as time-of-day normalization or trend analysis allow the RL agent to differentiate between expected daily fluctuations and genuinely suspicious behavior, such as traffic bursts that align with known attack schedules.

Content-based features delve into the specifics of packet content and header information. This includes identifying protocol anomalies, unusual header field values, payload characteristics (when available), and application-layer request patterns. Such features are crucial for detecting more sophisticated attacks that may not exhibit volume or distribution anomalies but instead exploit specific protocol vulnerabilities or payload structures.

A critical aspect of state space design is balancing informativeness with efficiency. High-dimensional state representations can slow down learning and lead to overfitting. To mitigate this, techniques like feature selection, dimensionality reduction, and hierarchical representations can be employed. These methods help retain the most relevant features while reducing computational overhead, ultimately enhancing the agent's learning performance and scalability.

2.3.2 Action Space Definition

The action space defines the set of all possible decisions or interventions that the reinforcement learning (RL) agent can make in response to observed network conditions. In the context of DoS detection systems, these actions are typically organized into three main categories: detection-related, response-related, and adaptive actions. Each category serves a distinct role in enabling the agent to not only identify malicious activity but also to react appropriately and adjust its behavior over time.

Detection-related actions allow the agent to classify the nature of incoming traffic. These may include flagging traffic as normal, marking it as a potential DoS attack with low confidence, confirming it as a high-confidence DoS event, or requesting additional inspection to reduce uncertainty. These actions contribute to the agent's ability to differentiate between benign and malicious behavior with varying degrees of certainty.

Response-related actions are used to enforce protective measures against suspicious or confirmed attack traffic. These actions include allowing traffic to proceed unimpeded, rate-limiting suspected malicious flows, blocking specific traffic patterns known to be harmful, or redirecting traffic for more detailed analysis by external systems. Such actions form the core of the system's active defense mechanism, enabling real-time mitigation of ongoing threats.

Adaptive actions provide the agent with the ability to adjust its operational parameters in response to evolving threat landscapes. This includes actions such as modifying monitoring sensitivity, tuning feature extraction processes, escalating suspicious cases to human analysts, or gathering additional contextual data to refine future decision-making. These adaptive strategies are essential for maintaining long-term performance in dynamic environments.

The granularity of the action space plays a critical role in determining both the learning efficiency and the practical utility of the RL system. An overly coarse action space may restrict the agent's responsiveness and lead to underfitting, whereas an excessively fine-grained space can increase the complexity of the learning problem and slow convergence. Therefore, careful design of the action space is essential to strike an optimal balance between expressiveness and tractability.

2.3.3 Reward Function Design

The reward function plays a pivotal role in shaping the learning behavior of the reinforcement learning (RL) agent, guiding it toward effective and efficient DoS detection strategies. A well-crafted reward function must strike a balance between multiple, often competing objectives such as detection accuracy, operational efficiency, and timely response.

In terms of detection accuracy, the agent is incentivized through positive rewards for correctly identifying attacks, while being penalized for false positives and false negatives. These penalties and rewards may be further weighted based on the severity of the detected attack and the agent's confidence in its classification. This ensures that the agent not only learns to detect attacks but also to assess their impact and act accordingly.

Operational efficiency is also integral to the reward function. Small penalties may be applied to account for resource consumption, including computational overhead and bandwidth usage. Additionally, time-based rewards can encourage early detection, while excessive inspection of normal traffic or inefficient use of mitigation mechanisms can incur penalties. These components help align the agent's behavior with real-world constraints and performance expectations.

A typical structure for the reward function may take the following form:

$$R(s, a, s') = w_1 \times \text{DetectionAccuracy} + w_2 \times \text{TimeToDetect} + w_3 \times \text{ResourceEfficiency} \tag{2.2}$$

In this formulation, *DetectionAccuracy* reflects the correctness of the classification, considering true positives, false positives, and false negatives. *TimeToDetect* quantifies the latency in identifying an attack, and *ResourceEfficiency* evaluates how judiciously the system utilizes its resources during detection and response. The weights w_1, w_2, w_3 are tunable parameters that allow the system to prioritize objectives based on organizational or operational goals.

Careful calibration of the reward function is essential to ensure that the agent develops policies that are not only accurate and robust but also practical within the resource constraints and real-time requirements of modern network environments.

2.3.4 Environment Dynamics

The environment dynamics define the mechanisms through which state transitions occur in response to the agent's actions. In the context of DoS detection, these dynamics are influenced by both the underlying traffic patterns and the mitigation strategies employed by the agent. State transitions are often governed by probabilistic rules that reflect how network traffic evolves over time. For instance, specific traffic behaviors may be more or less likely to follow certain patterns, such as bursty traffic or sustained high-volume flows indicative of an ongoing attack.

The agent's actions directly impact the state evolution. For example, blocking suspicious traffic may lead to a reduction in attack volume, altering the observable state characteristics. Similarly, redirecting traffic for analysis or applying rate limits can change flow distributions and entropy metrics within the network. It is also important to consider temporal aspects of the environment; the effects of certain actions may not be immediately visible. For instance, a rate-limiting policy might only manifest noticeable changes after a delay, as the network adapts or the attacker responds.

Modeling these dynamics accurately is essential for realistic simulation and effective training of reinforcement learning agents. It allows the agent to learn how its actions influence the environment and adjust its policy accordingly to achieve robust and adaptive DoS detection performance.

2.3.5 Challenges in Applying RL to Security Domains

While reinforcement learning (RL) holds significant potential for enhancing cybersecurity systems, its application to security contexts—particularly DoS detection—presents several unique challenges. One major hurdle is the high dimensionality of the state space. Network traffic generates vast and complex feature sets, requiring sophisticated representation and dimensionality reduction techniques to ensure tractable learning.

Another difficulty lies in the delayed nature of rewards. In many cases, the outcome of a security-related decision—whether it successfully prevented an attack or caused unintended side effects—may only

become clear after a considerable time delay. This complicates credit assignment and policy evaluation. Moreover, rewards in the security domain tend to be sparse; attack events are relatively rare in comparison to the continuous stream of benign network activity, making it difficult for the agent to gain useful feedback consistently.

Safety during the learning process is also a critical concern. Exploration, a core part of RL, must be managed carefully to avoid compromising system integrity. Unlike in traditional domains, erroneous actions in security systems can have severe consequences. Compounding this is the adversarial nature of the environment—attackers actively adapt their strategies to bypass detection mechanisms, forcing RL agents to learn in a moving target setting.

To address these challenges, structured educational frameworks and simulation environments can be employed. These offer controlled scenarios where RL techniques can be gradually introduced and refined before deployment in production systems.

2.3.6 Advantages of RL for DoS Detection

Despite the complexities involved, RL brings numerous advantages to the domain of DoS detection. One of the most notable is adaptability. Unlike rule-based systems that require manual updates, RL agents can dynamically adjust to novel attack strategies and traffic behaviors. This makes them particularly well-suited for rapidly evolving threat landscapes.

RL also enables contextual decision-making, where responses are informed by the broader network state rather than isolated events. This holistic approach helps reduce false positives and improve detection accuracy. Moreover, RL frameworks can be designed to balance multiple, sometimes conflicting objectives—such as maximizing detection rates while minimizing the impact on normal traffic—through multi-objective optimization.

A proactive defense posture is another key benefit. By interacting continuously with the environment, RL agents can learn to anticipate and preempt attack progression rather than merely reacting to already-occurred incidents. This ongoing interaction fosters continuous improvement, allowing the system to refine its policies over time and maintain effectiveness even as network conditions and attack techniques evolve.

Chapter 3

Architectural Blueprint of a DQN-Powered DDoS Detection System

3.1 Introduction

¡¡¡¡¡¡¡ HEAD

======= ¿¿¿¿¿¿¿ 44bc684 (add some changes) This chapter provides a detailed examination of the architectural design and implementation of a Distributed Denial of Service (DDoS) detection system leveraging a Deep Q-Network (DQN).

We will systematically deconstruct the entire pipeline, beginning with the foundational steps of dataset acquisition and preprocessing. From there, we explore the critical process of feature engineering required to transform raw network data into a format suitable for a neural network.

The core of our discussion will focus on the DQN model itself: its underlying neural network architecture, the Q-learning algorithm that drives its decision-making, and the mechanisms that enable it to learn and adapt to evolving threat landscapes.

3.2 Comprehensive Data Preprocessing Pipeline

¡¡¡¡¡¡¡ HEAD The primary dataset for this research is the CIC-DDoS2019, developed by the Canadian Institute for Cybersecurity (CIC). This dataset is exceptionally relevant as it captures a comprehensive set of contemporary DDoS attack vectors. It includes both reflection-based attacks that exploit UDP protocols (such as DNS, NTP, and SSDP amplification) and connection-based TCP floods (like SYN and ACK attacks). This diverse attack landscape is integrated with realistic, multi-protocol benign traffic profiles, including common application-layer protocols like HTTP, HTTPS, and FTP. This composition ensures the data closely simulates a real-world network environment, making it an ideal benchmark for evaluating modern intrusion detection systems.

The raw data, provided in PCAP format, was processed using the CICFlowMeter-V3 tool to generate labeled network flows. Each flow is uniquely identified by its 5-tuple (source/destination IP addresses, ports, and protocol). From these raw flows, CICFlowMeter extracts a rich set of over 80 statistical and time-based features. These include metrics such as flow duration, forward and backward packet counts, packet length statistics (min, max, mean), and flow I/O rates (bytes/sec). Crucially for supervised learning, each generated flow is explicitly labeled as either 'Benign' or with its specific attack type (e.g., 'DrDoS_NTP'), providing the ground truth for training our model.

This phase is paramount for ensuring the model's robustness and performance, as it involves meticulously cleaning and preparing the raw dataset. Our focus here is on retaining only high-quality, relevant samples,

thereby mitigating noise and irrelevant features that could lead to overfitting and reduced detection accuracy.
=======

3.2.1 Dataset

The primary dataset used for this research is **CICDDoS2019**. This dataset is highly relevant as it contains a variety of the most up-to-date common DDoS attacks alongside benign traffic, closely resembling real-world network scenarios. The data is provided in PCAP format and has been pre-processed using the CICFlowMeter-V3 tool to generate labeled flows in CSV files. Each flow is characterized by features based on its time stamp, source and destination IPs, source and destination ports, and the protocol used.

¿¿¿¿¿¿¿ 44bc684 (add some changes)

3.2.2 Step 1: Data Cleaning

The initial phase focuses on cleaning the dataset to ensure its quality and integrity. This involves three key actions:

1. **Handling Null Values:** The dataset is first inspected for any missing or null values. Rows containing such values are dropped entirely. This step is crucial to prevent computational errors during model training and to ensure that the model learns from complete, high-quality samples.
2. **Dropping Irrelevant Features:** Certain features in the dataset, such as `Flow ID`, `Source IP`, `Destination IP`, `Source Port`, and `Timestamp`, are removed. These features are unique to each specific flow and do not represent generalizable patterns of an attack. Including them would introduce noise and create a high risk of overfitting, where the model memorizes specific instances from the training data rather than learning the underlying behavior of attacks.
3. **Numeric Conversion:** All remaining feature values are converted to a numeric data type. Machine learning models, and particularly neural networks like the DQN, require numerical input for their mathematical operations. This step ensures that all data, including any encoded categorical features, is in a format that the model can process.

3.2.3 Step 2: Binary Label Transformation

To simplify the detection task, the problem is framed as a binary classification problem. The multi-class labels present in the original dataset are converted into a binary format:

- Traffic labeled as **'BENIGN'** is mapped to the value **0**.
- Traffic representing any type of attack (e.g., 'DrDoS_NTP', 'SYN', 'UDP-lag') is mapped to the value **1**.

This transformation allows the model to focus on the fundamental task of distinguishing any malicious activity from normal network behavior.

3.2.4 Step 3: Feature Selection using Random Forest Classifier

Feature selection is a critical process for reducing the dimensionality of the dataset, which in turn reduces model complexity, decreases training time, and mitigates the risk of overfitting. For this task, we utilize the **Random Forest Classifier**.

Random Forest is an ensemble learning method that constructs a multitude of decision trees during training. It provides a built-in measure of feature importance, which reflects the degree to which each feature contributes to improving the purity of the nodes in the trees. The process is as follows:

1. A Random Forest Classifier is trained on the cleaned dataset.
2. The `feature_importances_` attribute of the trained model is used to extract an importance score for each feature.

3. Features are ranked based on these scores, and only the top-ranked features that contribute most significantly to the model's predictive power are retained for subsequent steps.

3.2.5 Step 4: Data Normalization

After feature selection, the data is normalized to ensure that all features contribute equally to the model's learning process. We employ **Min-Max normalization**, which scales each feature to a fixed range between 0 and 1.

This technique is essential for neural networks, as it prevents features with larger numeric ranges from dominating the learning process and helps stabilize the gradient descent optimization, leading to faster convergence. The Min-Max normalization formula for a feature X is given by Equation 3.1.

$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{3.1}$$

where:

- $X_{\text{normalized}}$ is the scaled value.
- X is the original feature value.
- $X_{\min}$ is the minimum value of the feature in the dataset.
- $X_{\max}$ is the maximum value of the feature in the dataset.

3.2.6 Step 5: Dataset Balancing with Random Undersampling

The CICDDoS2019 dataset exhibits a significant class imbalance, with a much larger proportion of attack traffic compared to benign traffic. Training a model on such an imbalanced dataset would bias it towards the majority class (attacks), leading to poor detection performance for the minority class (benign traffic) and a high false positive rate.

To address this, we apply **random undersampling**. This technique balances the class distribution by reducing the number of samples from the majority class. Specifically, we randomly select and remove samples from the 'attack' class until its size matches the number of samples in the 'benign' class. While this method risks discarding potentially useful information from the majority class, it is effective in creating a balanced dataset that enables the model to learn the distinguishing characteristics of both classes with equal importance. The result is a model that is more robust and accurate in classifying both attack and benign traffic.

3.3 Applying Reinforcement Learning to Supervised Problems

This section addresses a core methodological challenge: adapting a paradigm designed for sequential decision-making—Reinforcement Learning (RL)—to a static, non-sequential classification task. The standard RL framework involves an agent learning through continuous interaction with a dynamic environment to maximize a cumulative reward. In contrast, our context for DDoS detection is defined by a finite, pre-labeled dataset where concepts like "sequence" and "environmental response" are not inherently present.

To fully appreciate the nature of this challenge and the elegance of the proposed solution, it is first necessary to contrast the foundational philosophies of the two machine learning paradigms at play: supervised learning and reinforcement learning.

3.3.1 The Methodological Dichotomy: Supervised vs. Reinforcement Learning

Most classification problems—like intrusion detection—are traditionally handled using supervised learning. In this setup, we train a model on a labeled dataset (X,Y)(X,Y), where XX is a collection of input features (such as network flow statistics), and YY contains the corresponding labels (e.g., "Benign" or "Attack").

The goal is to learn a function f:X→Yf:X→Y that can generalize well enough to predict labels for new, unseen inputs.

Training is guided by a loss function (like cross-entropy), which quantifies the error between the model's prediction f(x)f(x) and the true label yy. Optimization algorithms such as gradient descent are then used to minimize this error. In essence, supervised learning provides direct, explicit feedback — the model learns by being corrected.

Reinforcement Learning (RL), on the other hand, approaches learning very differently. Instead of learning from labeled examples, an RL agent learns through trial and error. At each time step, the agent observes the state of its environment, selects an action, and receives a scalar reward (or penalty). The goal isn't to reduce an immediate prediction error, but to learn a policy — a strategy for choosing actions that maximize long-term cumulative reward.

Unlike supervised learning, feedback in RL is often sparse and delayed. The signal comes not from correctness per se, but from the eventual outcome of the agent's decisions. This makes RL better suited to problems where success is defined by sequences of actions rather than single-point predictions.

So, here's the tension: Supervised learning is about learning from a teacher, whereas reinforcement learning is about learning from experience. Our dataset fits the supervised mold, but we're interested in applying RL techniques — especially those based on Deep Q-Networks (DQN) — which are typically designed for dynamic decision-making tasks.

3.3.2 Reframing the Problem: Classification as a Decision Process

To bridge this gap, we re-imagine the classification problem as a simplified decision-making process. Essentially, we simulate a one-step reinforcement learning (RL) environment from static data, enabling us to leverage Deep Q-Network (DQN) optimization.

Here's how we map the core RL elements to our classification setup:

- **State (s_t)**: Each state is a single, preprocessed feature vector from the dataset — essentially a snapshot of network traffic. Unlike in traditional RL, the state doesn't evolve from a prior action. To maintain the Q-learning flow, we define the "next state" (s_{t+1}) as the next sample in a randomly shuffled batch. While this doesn't reflect true temporal progression, it allows us to estimate future value.
- **Action (a_t)**: The action corresponds to choosing a class label. In our binary classification task, the agent can label a sample as either `Benign` or `Attack`. The learned policy $\pi(a \mid s)$ reflects the model's classification strategy.
- **Reward (r_t)**: Once an action is taken (i.e., a label is predicted), we compare it to the true label and assign a reward: for example, $+1$ for a correct classification and 0 or -1 for an incorrect one. This ties the RL feedback directly to supervised ground truth, giving the agent a clear sense of whether its decision was "good."

By aligning these components, we shift the learning goal. Instead of directly minimizing classification error, we ask the model to learn the *value* of choosing a particular label in a given context. This opens up the use of DQN, transforming the static task of classification into a process of value-based decision-making.

3.4 Deep Q-Network Agent: Operational Mechanics

Having established the conceptual mapping between reinforcement learning and supervised classification, this section details the operational mechanics of the Deep Q-Network (DQN) agent. We begin by providing a high-level, component-wise breakdown of the unified training loop that forms the core of our methodology. This framework is *architecturally agnostic*, meaning it governs the learning process regardless of whether the internal Q-network is a multilayer perceptron (MLP) or a convolutional neural network (CNN).

The primary goal of this training process is to iteratively refine the agent's Q-network, which is responsible for approximating the optimal action-value function:

$$Q^*(s, a)$$

This function estimates the long-term value of taking a specific action (i.e., making a classification) when presented with a particular state (i.e., a network traffic sample).

3.4.1 Component 1: Generic Sample and Data Provisioning

The learning process begins with the provisioning of data to the agent. At each training step, a *generic sample* is drawn from the preprocessed and shuffled dataset. This sample is not merely a single data point but a tuple containing three critical elements required for the Q-learning update:

$$(s_t, a_t^*, s_{t+1})$$

- **Current State (s_t)**: This is the feature vector of the network traffic sample currently under evaluation. It represents the agent's complete observation of the "environment" at the present moment.
- **Optimal Action (a_t^*)**: This is the ground-truth label corresponding to state s_t. In this framework, the true label is re-conceptualized as the "optimal action"—the decision a perfect, omniscient agent would make. This component injects the essential supervised signal into our RL framework.
- **Next State (s_{t+1})**: This is the feature vector of the subsequent sample in the shuffled dataset. As our data is non-sequential, s_{t+1} does not represent a temporal consequence of an action taken in s_t. Its role is purely functional: it serves as a lookahead mechanism, allowing the agent to estimate the value of the next state, which is a crucial element for calculating the target Q-value.

Algorithm 1 Data Sampling Step

```
// Retrieve a minibatch of N generic samples from the pre-structured dataset.
// Each sample is a tuple (s_t, a_t^*, s_{t+1}).
minibatch ← SampleFromDataset(dataset, batch_size = N)
// Unpack the minibatch into separate tensors for processing.
S_t, A_t^*, S_{t+1} ← unpack(minibatch)
// S_t:  Current state vectors.
// A_t^*:  Ground-truth label indices (optimal actions).
// S_{t+1}:  Next state vectors.
```

Pseudocode: Data Provisioning

3.4.2 Component 2: The ϵ-Greedy Policy and Action Selection

Once presented with the current state s_t, the agent must select an action a_t. This decision is governed by an ϵ-greedy policy—a fundamental strategy in reinforcement learning designed to manage the critical exploration-exploitation trade-off.

- **Exploitation**: With probability $(1 - \epsilon)$, the agent chooses to exploit its current knowledge. It feeds the state s_t into its Q-network, which predicts the Q-value for each possible action. The agent then greedily selects the action with the highest predicted Q-value:

  $$a_t = \arg\max_a Q(s_t, a)$$

  This represents the agent acting on what it currently believes to be the best strategy.
- **Exploration**: With probability ϵ, the agent explores. It disregards its learned policy and selects an action at random from the available action space. Exploration is essential for discovering potentially better strategies and avoiding premature convergence to suboptimal policies.

The parameter ϵ is a tunable hyperparameter that is typically annealed over the course of training. Initially set to a high value (e.g., 1.0) to encourage exploration when the agent is inexperienced, it gradually decays to a smaller value (e.g., 0.01) as the Q-network becomes more accurate.

Algorithm 2 Epsilon-Greedy Action Selection

```
function SelectAction(state, q_network, ϵ)
    p ← RANDOM_FLOAT(0, 1)
    if p < ϵ then
        // Exploration: choose random action
        return RANDOM_CHOICE(ACTION_SPACE)
    else
        // Exploitation: choose best action
        q_values ← Q_NETWORK.PREDICT(state)
        return ARGMAX(q_values)
    end if
end function
```

Pseudocode: Epsilon-Greedy Action Selection

3.4.3 Component 3: The Reward Function

After the agent selects and executes an action a_t, the simulated environment provides an immediate scalar reward r_t. This reward function serves as the primary feedback mechanism, quantitatively evaluating the quality of the agent's most recent decision. It forms the crucial bridge between the supervised ground-truth label and the reinforcement learning (RL) feedback loop.

The reward is calculated by directly comparing the agent's chosen action a_t with the known optimal action a_t^*. A typical implementation is:

$$r_t = \begin{cases} +1 & \text{if } a_t = a_t^* \quad \text{(Correct Classification)} \\ 0 & \text{if } a_t \neq a_t^* \quad \text{(Incorrect Classification)} \end{cases}$$

Using a neutral reward of 0 for incorrect actions, rather than a negative penalty (e.g., -1), can lead to more stable training in some contexts, as it does not excessively punish the agent during its initial exploration phase.

Algorithm 3 Reward Calculation

```
function GetReward(predicted_action, optimal_action)
    if predicted_action == optimal_action then
        return +1                          ▷ Positive reinforcement for correct decision
    else
        return 0                           ▷ Neutral feedback for incorrect decision
    end if
end function
```

Pseudocode: Reward Calculation

Component 4: The Target Q-Value (q_{ref}) Calculation

This step is the most sophisticated and innovative aspect of the entire training framework. In standard supervised learning, a model's output is compared against a static, ground-truth label. In our DQN framework, the Q-network is trained against a dynamically constructed hybrid target vector, q_{ref}. This target ingeniously blends the explicit signal from the supervised label with the value-estimation principles of reinforcement learning.

The construction of this target vector for the current state s_t proceeds as follows:

- **Estimate Optimal Future Value:** The Q-network is used to predict the Q-values for the next state s_{t+1}. The maximum of these values is taken as the best estimate of the optimal value achievable from the next state:
$$\max_a Q(s_{t+1}, a).$$
- **Initialize Target Vector:** A target vector q_{ref}, of the same dimension as the action space, is initialized with zeros:
$$q_{\text{ref}} = \vec{0}.$$
- **Inject the Supervised Signal:** A value of 1.0 is placed in the target vector at the index corresponding to the true label a_t^*. This is the most powerful signal, directly anchoring the learning process to the ground truth:
$$q_{\text{ref}}[a_t^*] = 1.0.$$
- **Inject the Reinforcement Learning Signal:** The estimated optimal future value is discounted by a factor λ and added to the index of the agent's chosen action a_t:
$$q_{\text{ref}}[a_t] += \lambda \cdot \max_a Q(s_{t+1}, a).$$

This dual-signal approach ensures the agent learns not only to identify the correct answer but also to accurately value its own decisions in the context of future possibilities.

Algorithm 4 Target Q-Value Calculation

```
function CalculateTarget(q_network, S_t, S_{t+1}, A*_t, A_t, λ)
    q_next ← q_network.predict(S_{t+1})
    max_q_next ← max(q_next, axis = 1)
    q_ref ← zeros_like(q_network.predict(S_t))
    for all i in batch do
        true_label ← A*_t[i]
        agent_action ← A_t[i]
        discounted_future ← λ · max_q_next[i]
        q_ref[i][true_label] ← 1.0
        q_ref[i][agent_action] += discounted_future
    end for
    return q_ref
end function
```

Pseudocode: Target Q-Value Calculation

Component 5: Loss Calculation and Network Update

The final step in the training loop is to update the Q-network's parameters (weights and biases) to improve its predictions. This is achieved by quantifying the error between the network's output and the hybrid target, and then adjusting the weights to minimize this error.

- **Final Target Preparation:** The hybrid target vector, q_{ref}, may contain multiple non-zero values with continuous components. To make it compatible with a classification-style loss function, it is first converted into a definitive one-hot vector. This is done by finding the index of the maximum value in q_{ref}:
$$i^* = \arg\max_i q_{\text{ref}}[i],$$
and setting the one-hot vector $\hat{y}$ such that:
$$\hat{y}[i] = \begin{cases} 1 & \text{if } i = i^*, \\ 0 & \text{otherwise.} \end{cases}$$

- **Loss Function:** The discrepancy between the Q-network's predicted Q-values (after applying a Softmax function to obtain a probability distribution) and the one-hot target vector $\hat{y}$ is computed using the Cross-Entropy Loss:
$$\mathcal{L} = -\sum_i \hat{y}[i] \cdot \log(\text{Softmax}(Q(s_t))[i]).$$
- **Optimization:** The computed loss $\mathcal{L}$ is used to guide the optimization process. An optimizer such as Adam is employed to compute the gradients of the loss with respect to the network's parameters.
- **Backpropagation:** These gradients are propagated backward through the layers of the Q-network using the backpropagation algorithm, from the output layer toward the input.
- **Weight Update:** The optimizer then updates the parameters of the Q-network in the direction that minimizes the loss. This parameter update is denoted by:
$$Q^{(t)} \to Q^{(t+1)}.$$

This complete update cycle forms a single iteration of the learning process. Repeating this process over many training samples and epochs enables the Q-network to gradually converge to an optimal policy, effectively learning to estimate the value of each classification choice for any given network traffic profile.

The architectural details of the Q-network will be discussed in the following section.

3.4.4 Architectural Implementation I: The Multi-Layer Perceptron (MLP)

The first architectural implementation for our Q-network is a **Multi-Layer Perceptron (MLP)**, a standard type of feedforward neural network. This model acts as a powerful function approximator, processing the input state vector in its raw, one-dimensional form. MLPs are particularly well-suited for data without explicit spatial or temporal structure—ideal for tabular or vectorized representations.

This subsection deconstructs the MLP architecture layer by layer. Each component—from input to output—is examined both structurally and conceptually to explain not just how it works, but why it is designed this way.

Input Layer and Normalization. The network begins with an input layer that receives a mini-batch of state vectors S_t, where each vector has a dimensionality equal to the number of input features in the dataset.

To stabilize learning, the input passes through a **1D Batch Normalization** layer, which addresses the issue of *internal covariate shift*. For each activation x_k in the mini-batch, the normalized output $\hat{x}_k$ is computed as:

$$\hat{x}_k = \frac{x_k - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

where μ_B and σ_B^2 are the mini-batch mean and variance, and ε is a small constant for numerical stability.

This normalization is followed by a learned affine transformation that restores representational capacity. By ensuring consistent feature distributions, it prevents features with large scales from dominating and accelerates convergence.

First Hidden Layer. Next, the normalized input is passed into a fully connected (linear) layer with 256 output units. Each neuron in this layer computes a weighted sum of its inputs, adds a bias term, and prepares the output for non-linear activation.

The use of 256 units offers a high-capacity representation that can extract a broad range of low-level features from the raw input.

ReLU Activation. The layer's outputs are processed through the ReLU activation function:

$$\text{ReLU}(x) = \max(0, x)$$

This introduces non-linearity, enabling the network to model complex patterns. ReLU also mitigates the vanishing gradient problem, is computationally efficient, and promotes sparsity in the activations.

Intermediate Block: BatchNorm, Dropout, Linear. The output from the first hidden layer enters a second block consisting of:

- A **Batch Normalization** layer to re-stabilize the distribution of features.
- A **Dropout** layer with a rate of 0.3, which randomly sets 30% of the units to zero during training, forcing the network to learn more robust representations.
- A **Linear layer** that reduces the dimensionality from 256 to 128, forming a representational bottleneck that encourages information compression and abstraction.

This "funnel" structure encourages the network to retain only the most salient features, enhancing generalization.

Output Layer and Softmax. The final output of the hidden layers is passed through a linear layer that maps the 128-dimensional feature vector to the number of discrete actions (e.g., 2 in a binary setting), producing raw logits $z = (z_1, z_2, \ldots, z_K)$.

A **Softmax function** is applied to convert these logits into a probability distribution:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

This ensures all output probabilities are non-negative and sum to 1, facilitating training with the cross-entropy loss.

This MLP-based Q-network is a structured stack of linear transformations, ReLU activations, batch normalization, and dropout layers. It incrementally transforms the raw input into increasingly abstract and robust features, ultimately yielding well-calibrated Q-value predictions for each action in the environment.

3.4.5 Architectural Implementation II: The Convolutional Neural Network (CNN)

The second architectural implementation for our Q-network leverages a **Convolutional Neural Network (CNN)**. This represents a more sophisticated approach, predicated on the hypothesis that meaningful, localized patterns and spatial correlations exist within the network traffic feature set. Unlike the MLP, which treats features as an unstructured vector, the CNN is designed to exploit the structure of its input, making it a powerful tool for automatic and hierarchical feature extraction.

This subsection provides a detailed deconstruction of the CNN-based Q-network. We begin by explaining the critical pre-processing step of transforming the one-dimensional feature vector into a two-dimensional, image-like grid. We then analyze the function and rationale of each specialized layer—convolutional, pooling, and dropout—before examining how these components are integrated with fully connected layers to produce the final Q-value estimations.

The Foundational Step: Transforming Tabular Data into a Feature-Image

The core premise of this approach is the transformation of the one-dimensional state vector into a two-dimensional grid, or *feature-image*. This process reframes the problem from numerical analysis into a form of image recognition, enabling the application of convolutional networks.

Mechanism of Transformation. A state vector containing N features is reshaped into a grid of height H and width W, such that $H \times W = N$. For example, a 1x64 state vector can be reshaped into an 8x8 grid, where each row corresponds to 8 features:

- Features 1–8 form the first row.
- Features 9–16 form the second row.
- ...
- Features 57–64 form the eighth row.

This results in a single-channel, 8x8 feature-image where each "pixel" corresponds to a normalized feature value.

The Critical Importance of Feature Ordering. The order in which features are placed into the grid significantly impacts the patterns that the CNN can detect. Strategies include:

- **Default Ordering:** Uses the order from the feature extraction tool.
- **Correlation-Based Ordering:** Places highly correlated features next to each other using a correlation matrix.
- **Domain Knowledge-Based Ordering:** Groups features by their semantic role (e.g., timing, packet size).

This ensures that local spatial regions in the grid carry meaningful information for pattern detection.

The Underlying Hypothesis. This approach tests the hypothesis that localized interactions between features can be detected more effectively by CNNs, leveraging their capacity for local pattern recognition.

Architectural Deep Dive

Input Normalization. A 2D Batch Normalization layer normalizes the feature-image, ensuring stable training.

First Convolutional Block: Low-Level Pattern Detection.

- **Convolutional Layer:** Applies 16 learnable 3x3 kernels to detect localized patterns.
- **ReLU Activation:** Introduces non-linearity.
- **Max Pooling:** Reduces spatial resolution, introduces translation invariance.
- **Spatial Dropout:** Applies 2D dropout with a rate of 0.25 to prevent co-adaptation.

Second Convolutional Block: Hierarchical Feature Abstraction.

- **Convolutional Layer:** Uses 32 3x3 kernels to detect higher-level features.
- **ReLU Activation and Dropout:** Follow the same structure to enhance abstraction and prevent overfitting.

The Fully Connected Head: Q-Value Estimation.

- **Flattening:** Converts the multi-dimensional tensor into a one-dimensional vector.
- **Dense Layer:** A fully connected layer with 64 neurons, followed by BatchNorm and ReLU.
- **Dropout:** A high-rate (0.5) dropout layer adds regularization.
- **Output Layer:** A linear layer maps to the number of possible actions, producing raw Q-value logits.

The final Q-values are suitable for training with a loss function such as Cross-Entropy Loss, and a softmax activation may be applied implicitly during optimization.

Bibliography

[1] IBM, *Security Fundamentals?*,
https://www.linkedin.com/pulse/fundamentals-security-slammghana/

[2] CIA, *Principles of Information Security.*
Available at: https://www.cleanpng.com/png-information-security-confidentiality-availability-1373941/

[3] Fidelis Security, *Understanding DoS and DDoS Attacks*,
https://fidelissecurity.com/blog/dos-ddos-attacks

[4] Imperva, *Denial of Service (DoS) Attack*,
https://www.imperva.com/learn/ddos/denial-of-service/

[5] NETSCOUT, *Slowloris Attack*,
https://www.netscout.com/blog/asert/slowloris-attack

[6] MDPI, *Anomaly-based Detection of DDoS Attacks Using Machine Learning*,
https://www.mdpi.com/2076-3417/10/3/1052

[7] IBM, *What is an Intrusion Detection System (IDS)?*,
https://www.ibm.com/think/topics/intrusion-detection-system

[8] Wikipedia, *Intrusion Detection System*,
https://en.wikipedia.org/wiki/Intrusion_detection_system

[9] Fortinet, *What is Intrusion Detection Systems (IDS)? How does it Work?*,
https://www.fortinet.com/resources/cyberglossary/intrusion-detection-system

[10] TechTarget, *What Is an Intrusion Detection System (IDS)?*,
https://www.techtarget.com/searchsecurity/definition/intrusion-detection-system

[11] TechTarget, *Dataset Difinition*,
https://www.unb.ca/cic/datasets/ddos-2019.html