

A NOVEL DEEP LEARNING APPROACH USING AUTOENCODER-BASED FEATURE LEARNING FOR CROSS-SITE SCRIPTING DETECTION

Imtiaz Ali Khoso^{*1}, Anika Joyo², Ghulam Yasin Khoso³, Muhammad Rafiue⁴,
Muzamil Sabir Bozdar⁵, Nadar Hussain⁶

^{*1,2,3,4,5,6}Department Information Technology Centre, Faculty of Agricultural Social Sciences, Sindh Agriculture University, Tandojam, Pakistan

¹imtiazali78699@gmail.com, ²anikajoyo5897@gmail.com, ³ghulamyasinkhoso914@gmail.com,
⁴muhammadrafiuemastoi@gmail.com, ⁵bozdarsahab263@gmail.com, ⁶nadarhussain02@gmail.com

DOI: <https://doi.org/10.5281/zenodo.22009494>

Keywords

Cross-Site Scripting (XSS), Machine Learning, Deep Learning, Web Security, Intrusion Detection System, RNN, LSTM, Autoencoder, ANN, Support Vector Machine

Article History

Received: 16 June 2026
Accepted: 12 August 2026
Published: 17 August 2026

Copyright @Author

Corresponding Author: *
Imtiaz Ali Khoso

Abstract

Client and server-side scripting plays an important role in contemporary web applications to deliver interactive content and customize it along with attracting more users. The same scripting power, however, provides an opportunity for Cross-Site Scripting (XSS), a malicious attack where the attacker embeds malicious script in pages to which the visitor has already given trust which is then used to harvest sensitive data. This mechanism is used by threat actors, either directly or via a set of intermediaries to attack normal Internet users. Usual targets are to gain access to the remote system, capture user's login information, access personal records, steal session cookies and deliver malware; stealing cookies is a common secondary motive. XSS is one of the most common and dangerous web application attacks and is also one of the OWASP Top 10 (2021) attacks. In 2023, XSS is the most critical most vulnerability with 19.10% of the cases, reported Alhamyani and Alshammari (2024). The present study compares a set of classifiers to detect XSS: Logistic Regression, Gradient Boosting, Naive Bayes, K-Means, Random Forest, Artificial Neural Network and Support Vector Machine. The accuracy of the model was the highest for the ANN (98.98%) model, which shows the significance of the deep-learning based method for separating malicious traffic from benign traffic.

1. INTRODUCTION

1.1 Overview of Cross-Site Scripting Attacks

As features and functionality have become dynamic and script-driven on social sites, cloud services and e-commerce platforms, the way modern web apps are used has changed and so have the opportunities for attackers. With the increased dependency on the web-based services, threats to user data, session details and application integrity have been increased and Cross-Site Scripting (XSS) is one of the most prevalent and harmful threats (Kaur et al., 2023).

Often, security is an afterthought for the developers of such applications and this makes them vulnerable over and over again for attackers to exploiting them with ever-increasing methods. As indicated in Alhamyani and Alshammari (2024), XSS accounted for 19.10% of the worst web vulnerabilities that were reported in 2023 based on the EdgeScan 2023 Vulnerability Statistics Report. This paper will concentrate specifically on the detection of such attacks using machine learning methods. XSS is a type of vulnerability in any application that takes input from the user and lets an attacker put code into the user's browser that the other user can see, which the other user's browser will then run. Because it can be used to circumvent protections such as the Same-Origin Policy (SOP), XSS is now categorized under A03:2021 Injection in the OWASP Top 10 (2021), reflecting its essential nature as a JavaScript injection carried out within the victim's own browser session.

How damaging a given XSS incident turns out to be depends largely on the sensitivity of the data the targeted site handles and on how well its administrators have hardened it beforehand outcomes can range from a minor nuisance to a severe breach. The underlying weaknesses that XSS exploits typically already exist within the web software, hosting servers, or plug-ins a site depends on and these gaps are generally well documented. Once it's penetrated, malicious code can be injected into the content a page was intended to deliver to visitors; if the browser does not know the content was corrupted, the browser assumes it came from a legitimate source and acts accordingly, executing the malicious code with

whatever privileges the source had. This enables an attacker to access very sensitive assets, such as session cookies and other data stored in the browser, via indirect injection points. Therefore, it could be considered as one type of code injection attack in the larger category of code injection attacks (He et al., 2017).

1.2 Machine learning

Machine learning has become a common tool for spotting and containing XSS activity, largely because it tends to outperform traditional detection methods on both speed and accuracy. Prior work has favored classifiers such as Logistic Regression, Naive Bayes and SVM for this task (Chopra et al., 2021), while more recent deep-learning extensions of these methods have demonstrated strong potential for picking up complex XSS attack signatures across varied network settings (Vinayakumar et al., 2018).

1.3 XSS attacks detection based on Machine learning

XSS attacks take advantage of vulnerabilities in a Web application that allows for the insertion of code into Web pages that are then displayed to the user on the client's browser. Typically, there are three types of them: Stored (Persistent), Reflected (Non-Persistent), DOM-based (done on the client code). These scripts can be used after they are installed to steal the session cookies, log keystrokes, steal identities or even be used to deliver malware. It has been shown that machine learning techniques can be used to accurately classify malicious traffic from normal web traffic (Alhamyani & Alshammari, 2024).

XSS is one of the most persistent and prevalent Web-application attacks based on the number of attacks and longevity. It usually happens in two phases: firstly, the attacker finds a vulnerable input and writes a malicious payload and secondly the victim executes that payload within their browser. The conventional approach to reducing the risk of these threats is to deploy a firewall and an Intrusion Detection System (IDS), which is a technology that can identify network traffic anomalies and unexpected behavior from an expected pattern (Kasim, 2020). The amount of

cybercrime incidents between 2015-2017 shows that it has been increasing steadily and there has been a huge amount of research into faster and more reliable methods of detection, of which machine learning is one of the best. The proliferation of deep-learning techniques and GPUs and the introduction of software-defined networking (SDN) has extended the potential of this area and has also given another option for isolating and managing XSS threats (Khuphiran et al., 2018).

1.4 Problem Statement

Signature or fixed rule based XSS detection systems are prone to producing lots of false positives and failing to stay current with the latest forms of attack variants. Machine-learning based alternatives have been shown to be able to detect malware and suspicious traffic but have their own challenges, including imbalanced training sets, limited feature set to capture attack behavior and limited generalizability across XSS variants. With the growing complexities of web applications and the sophistication of the modern attacks, there is still a need for detection frameworks that can detect known and new XSS patterns (Kaur et al., 2023; Alhamyani & Alshammari, 2024).

1.5 Objectives

1. To detect and understand the characteristics of a normal Web page as opposed to a page containing a cross-site scripting payload.
2. To develop and test a machine-learning approach to the XSS attack detection.

2. MATERIALS AND METHODS

2.1 System Detection

The goal of computing system design is to maintain the security of the data confidentiality, integrity, availability and to prevent service disruption. As networks, the Internet in particular, continue to grow, they expose more susceptible systems, allowing more systems to be susceptible to attack. The Morris ("Web") Worm in 1988; the Intersecting Set attack in 2004; the Slammer attack in 2005 and the Mirai attack in 2016 are just some of the many historical incidents that have demonstrated the potential for mass

disruption via flaws in an operating system or in applications that run on it.

The key to making a difference is being aware of a compromised device in the first place and that is why an Intrusion Detection System (IDS) is needed. An IDS does not attempt to prevent an attack from happening but rather reports to the administrator if an attack has occurred, making it a sort of "sensor" of suspicious activities rather than an "enforcer," who directly prevents an attack (Elsayed et al., 2020).

2.2 IDS Classification

There are three types of interruptions that can make the normal operation of a system non-executable:

- Breaches or violations tried vs. typical activity as documented baseline.
- Repeated patterns that relate to known exploits and/or previously identified weaknesses.
- Denial of service conditions (signatures of resource usage).

In brief, malicious use can be detected in most cases by behavior that is inconsistent with known identity schemas, breaches of known security boundaries, or exercise of unusual privileges (Sundaram, 1996).

2.3 Approaches of XSS Detection

Approaches to intrusion detection generally fall into two broad categories:

- Anomaly detection
- Misuse detection

2.3.1 Identification of Anomaly

The underlying idea is to establish a "normal activity profile" for a system and then treat any statistically significant departure from that baseline as a potential intrusion attempt. Two outcomes are worth weighing when building such a profile:

- Behavior that deviates from the norm without being overtly suspicious may still get labeled as an intrusion (a false positive).
- Conversely, genuinely malicious activity that doesn't look atypical can slip through undetected, a false negative, which is generally

considered the more dangerous of the two outcomes.

Anomaly-based systems (Figure 2.1) live or die by how well they set detection thresholds set them too aggressively and false positives spike; too leniently and false negatives slip through undetected. Deciding which behaviors merit

tracking in the first place is an equally critical design choice. These systems also tend to be computationally costly, since maintaining and continuously checking a large number of behavioral profile indicators demands significant processing overhead.

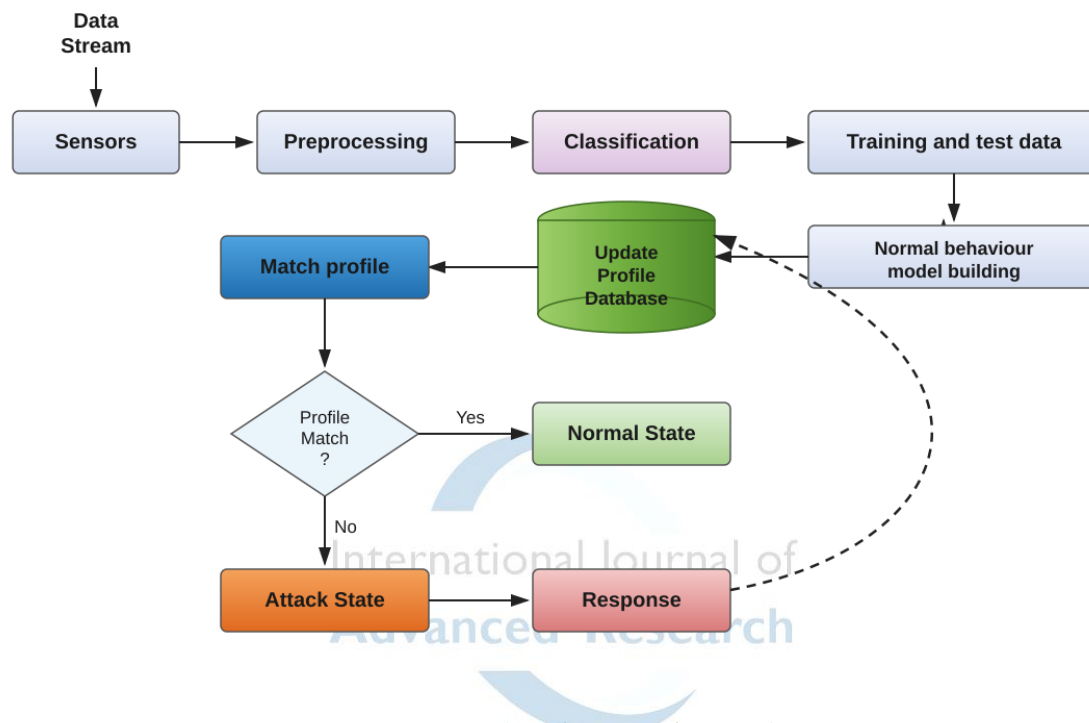


Figure 2.1 Identification of anomaly

2.3.2 Identification of network errors

In contrast, misuse-detection methods rely on known attack signatures to detect not only an exact attack, but variants of the attack, too (Figure 2.2). This is the way that a traditional antivirus system operates: it's capable of detecting any type of attack that has been seen before and added to

the repository pattern, but it doesn't provide protection against actual novel techniques. The central design challenge is crafting a signature broad enough to cover every variant of a given attack while narrow enough that it doesn't misfire on legitimate traffic.

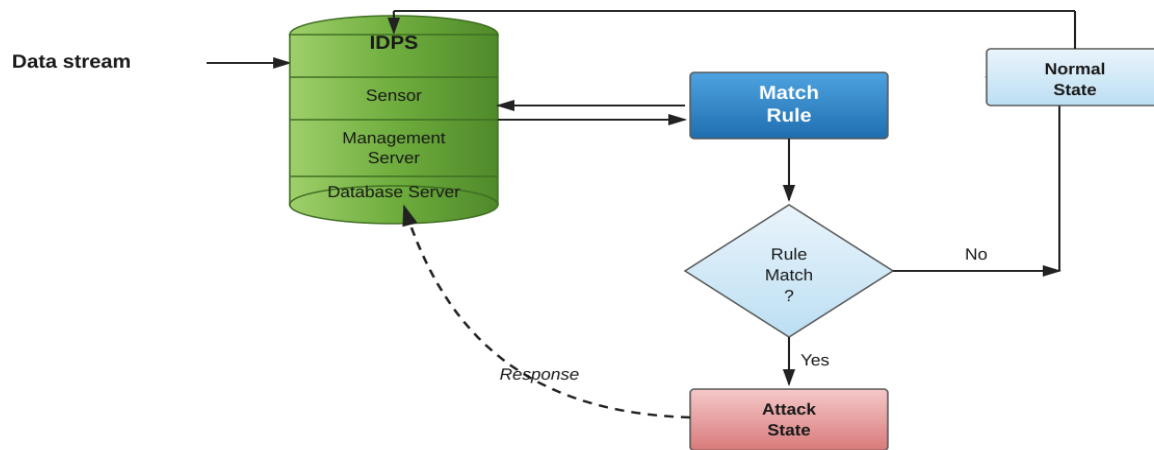


Figure 2.2 Identification of error or misuse system

2.4 IDS Data Sources and Extraction

The raw inputs an IDS uses when trying to detect malicious activity are sometimes called its data sources and fall into two general classification groups:

1. Host-based IDS
2. Network-based IDS

2.4.1 Host-based IDS

A host-based IDS is well suited to catching insider threats that don't necessarily generate network traffic (Figure 2.3), such as an unauthorized attempt to remotely log in and reach confidential information stored on a machine.

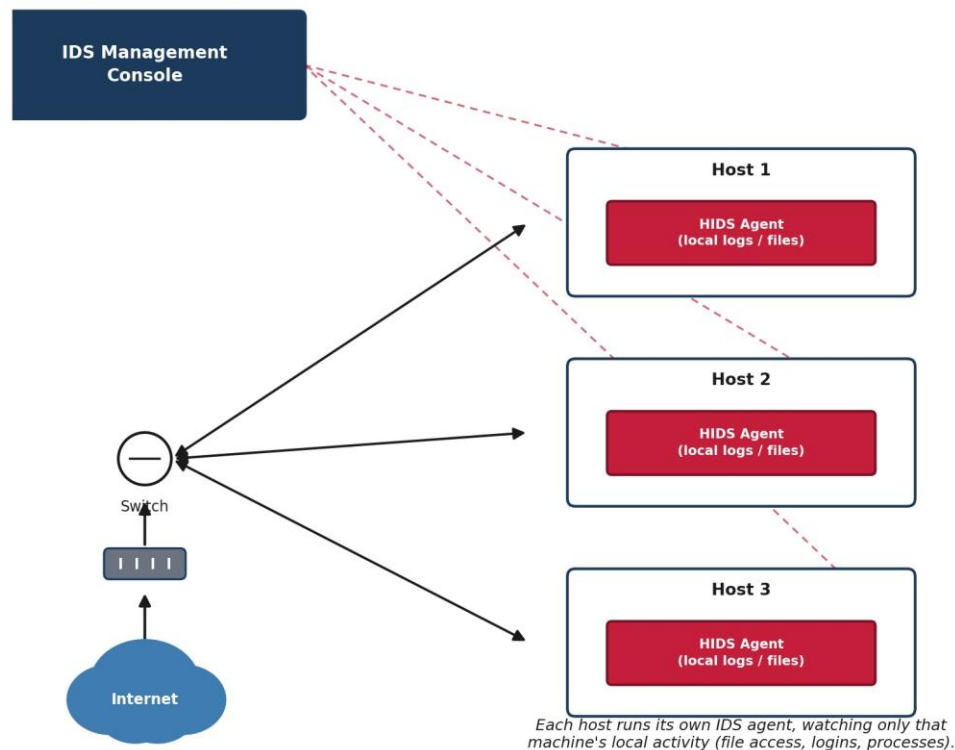


Figure 2.3 Host-based IDS framework

2.4.2 Network-based IDS Utilization

The network-based IDS, on the other hand, monitors traffic at the network border, using packet captures, NetFlow records and other sources of data to monitor multiple hosts

simultaneously (Figure 2.4). It can detect an attack launched from the outside network, at the center, while it is still in its initial stages, before it can penetrate to other machines (Khraisat et al., 2019).

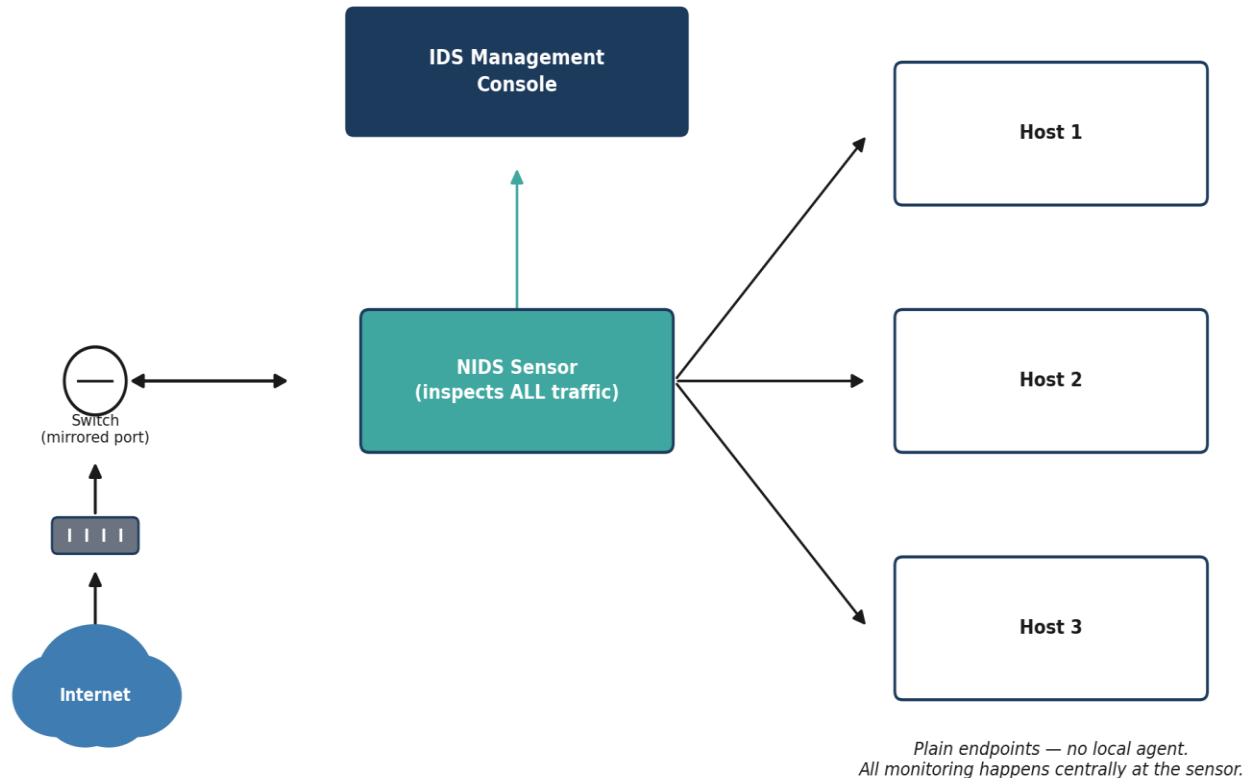


Figure 2.4 Network-based IDS framework

2.5 Utilization of SDN

Industry usage of the term SDN has drifted somewhat from its original, precise definition, now loosely applied to almost any networking approach built around software (Figure 2.5). Here, we adopt a narrower framing suited to address today's network challenges. Four defining characteristics distinguish SDN as an architecture:

- Control and data functions are split into separate entities, stripping network devices down to simple packet-forwarding roles.
- Forwarding decisions are guided by flow-level policy rather than destination alone; routers,

switches, firewalls and middleboxes can all coordinate through this flow abstraction, though flexibility is ultimately bound by what the available flow tables support.

- Control logic is centralized in the SDN controller's network operating system (NOS) software running on commodity server hardware that supplies the abstractions needed to simplify network design.
- The network becomes programmable through applications that interface directly with this core system, which is both a defining feature of SDN and its main selling point.

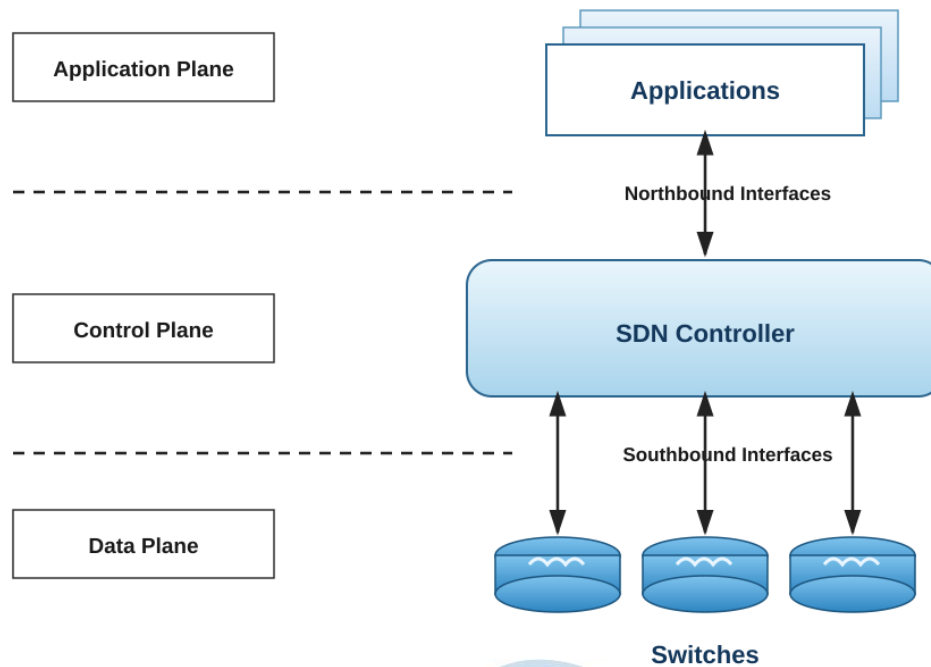


Figure 2.5 Proposed framework

2.5.1 Initial Architecture

SDN architecture is generally described in terms of three layers. The management plane consists of software that draws on both the northbound interface and the control plane to define control functions and set network-wide policy through dedicated control stations (Kreutz et al., 2014). The control plane carries out this logic, translating an application's high-level policy directives into instructions for the forwarding devices; because it works from a single, unified view of network state, applications built on it need not track topology themselves. The data plane, meanwhile, comprises the forwarding devices and the links connecting them and is programmed by the control plane through a southbound interface OpenFlow being the most widely adopted implementation of that interface (Kreutz et al., 2014).

2.5.2 SDN Utilization with IDS

One notable implication of this architecture is that a single, centralized controller can oversee an

entire network, which carries several practical consequences:

- Centralized control tends to simplify network management overall enhancements or changes become easier to roll out, which in turn reduces the frequency of human error (Elsayed et al., 2020) and IT teams are no longer tied to a single equipment vendor when deploying or updating infrastructure.
- Because the controller holds a consolidated view of the entire network, it effectively has a global picture of everything happening across it.
- Developers working in a simulated environment can take advantage of the SDN framework to deploy a range of applications that carry out network functions.
- Since the underlying infrastructure devices require no programming language of their own, operational costs fall well below those of a traditional network a major factor behind SDN's continued market growth.

That said, SDN's benefits come with a security trade-off: because the controller holds so much authority over the network, an intruder who compromises it puts the entire system at risk. This makes detecting anomalies including XSS activity within SDN traffic a critical part of the overall network design.

2.6 Utilization of Machine Learning (ML)

Deep learning (DL) is one of the branches of machine learning based on multi-layered neural networks, which learn hierarchies of feature representations directly from raw data without any feature engineering. The fact that DL is combined with supervised and unsupervised approaches for progressively building up the abstract representations from features learned from earlier layers makes it appropriate to capture the sophisticated behavior patterns found in XSS attacks (Shone et al., 2018).

2.6.1 Features auto encoder

The autoencoder is a deep learning model for unsupervised feature learning. The method involves compressing the input into a compact latent representation and reconstructing the original input from the compact representation, thereby forcing the network to learn efficient, information-dense encodings on the way (Shone et al., 2018).

Training is done via standard backpropagation, but the output of the network is just the input itself instead of an external label, which causes the network to learn to reproduce what it is given, not predicting some labels. This can feel counterintuitive at first, since there's no separate outcome being predicted the objective is purely reconstruction accuracy. The basic autoencoder consists of three layers: an input layer, a hidden (encoding) layer and an output layer which reconstructs the input layer. The hidden layer contains fewer dimensions than the input layer, so the network has no other option but to compress the data into a smaller representation to reconstruct at the output layer and it is this compression that compels the network to learn the most critical structure in the data (Mohammed et al., 2018).

2.6.2 Configuration of Stacked Autoencoder

Adding multiple autoencoders in a stack gives more depth to the network and each successive hidden layer learns a more abstract concept than the previous layer, in this fashion. This increases the depth of the model, which reduces the cost of training and the amount of data needed and also results in more accurate predictions. Each layer's output is directly used as input to the next layer: the first hidden layer usually captures first-order features, the second layer captures second-order patterns from the first layer and subsequent layers continue to capture higher-order patterns from the previous layers (Mohammed et al., 2018).

3. RESULTS

3.1 Proposed Methodology Approach

In classical machine learning, the relevant input features are typically hand-engineered and the algorithm's job is simply to map those features into an output. Deep architectures instead build up properties automatically across successive layers, with each layer's representation shaped by and building on, what the previous layer discovered.

This work draws on the XSS-attacks dataset published by Saurabh Shahane on Kaggle. In line with earlier studies in this area, our main goal is to tackle the classification problem using 10-fold cross-validation to evaluate the proposed detection model.

3.1.1 Proposed ML Classifications

Classification, in this context, means assigning each instance to its correct class label based on its measured features. While the same underlying technique can support both misuse and anomaly detection, it is more commonly applied to the former sorting the dataset into predefined categories.

3.2 Extraction and Preprocessing

Our focus here is on classification problems that help identify network attacks and distinguish between their different forms; any pattern tied to a specific attack type is treated as anomalous activity. Before the detection model can be trained, the dataset first needs to be properly prepared.

3.2.1 Extraction and Normalization

Taken together, the following observations about the dataset are worth noting (Karan et al., 2018):

- The dataset, delivered in socket format, originally carries fields such as Source IP, Destination IP and Stream ID. To prevent the model from overfitting to network-specific identifiers that wouldn't generalize, we removed all socket-level fields, leaving 77 distinct features besides the traffic-class label. Given the scale of the full training and testing sets, only a representative sample was used.

3.3 Experimental configurations

The detection method proposed here relies on a recurrent neural network (RNN) flag irregularities that conventional systems tend to miss, with training aimed at minimizing loss while keeping throughput high. We selected an autoencoder for

the anomaly-detection component specifically because it learns the dominant variables needed to reconstruct its input at the output layer. Combined with LSTM units, the model uncovers underlying patterns in the network dataset through a semi-supervised process.

Both the encoder and decoder stages are built from a substantial number of LSTM units. The encoder maps the input X_t into Z_t , a fixed-length feature vector, with the raw input taking the form $X_t \in \mathbb{R}^{77 \times 1}$. Moving through the decoder, filter dimensions expand across four successive layers 16, 32, 64 and 128, respectively (Figure 3.1). A SoftMax layer, configured with two output channels, converts the decoder's output into class probabilities, splitting the result into normal and attack categories. Each class receives a probability score between 0 and 1, alongside an overall probability estimate.

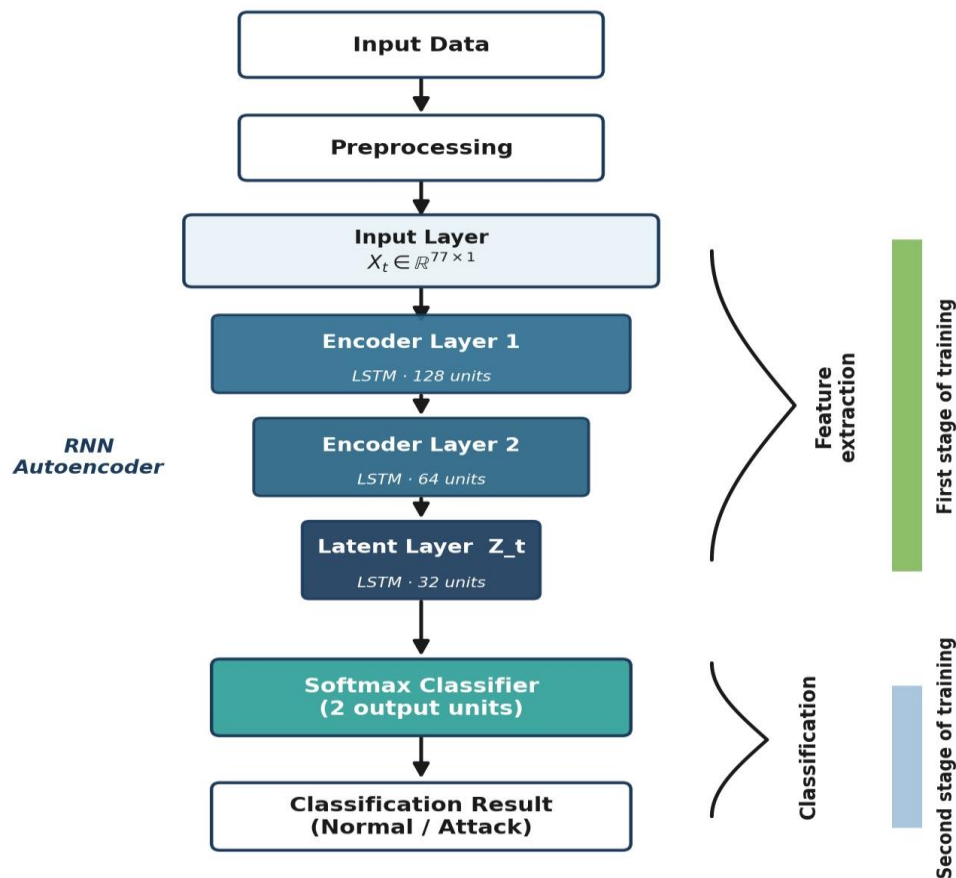


Figure 3.1 Machine learning based proposed framework model

3.4 Proposed data normalization

We evaluated our classifiers against a recently released XSS-attacks dataset covering attacks executed over application-layer protocols on both TCP and UDP, spanning two broad categories: exploitation-based and reflection-based attacks. Selecting the right features for an intrusion-detection system is inherently difficult and typically requires domain expertise, since attack patterns evolve constantly, making it unrealistic to hand-pick features tailored to any single attack type (Hijazi et al., 2018).

The dataset was prepared for training in flow-based format, with CIC Flow Meter having originally extracted over 80 features. After discarding the unneeded fields discussed above, 77 features remained as model input. We framed the task as binary classification, training the model to separate genuine from malicious traffic treating every XSS variant as an attack instance alongside the normal-traffic class.

3.5 Proposed Work

To extend the work, we propose investigating Andrews curves as visualization techniques of such high-dimensional feature space. It is a technique that maps multivariate data into two dimensions using a Fourier-series representation to identify clusters, patterns and outliers. Applying this to XSS feature data could improve how interpretable the classification results are and help guide feature selection in future models.

3.6 Validation of Receiver Operating Characteristic (ROC)

A Receiver Operating Characteristic (ROC) is a curve that illustrates the performance of a binary classifier as its decision to threshold changes. It was designed for analyzing radar signals and displays True Positive Rate versus False Positive Rate over a spectrum of thresholds and provides a threshold independent performance view of a classifier. In our evaluation, positive-class samples were used to compute TPR and negative-class samples to establish the FPR baseline, giving an overall picture of how well the proposed approach performs.

3.7 Results obtained by Fully featured utilization

Table 3.1 reports outcomes for every classifier tested against the fully featured version of the dataset. Across the board, scores for the XSS class come out considerably stronger than those for the normal class, a pattern that generally holds because XSS traffic tends to diverge more sharply from typical activity than normal traffic diverges from itself. Prior to training, we stripped out several socket-level fields source IP, destination port, destination IP, stream ID and date since these vary from network to network and would otherwise bias the deep-learning model toward memorizing socket identifiers rather than learning genuine packet-level attack characteristics, which would show up as overfitting. Among the seven classifiers, Logistic Regression and Support Vector Machine delivered the strongest precision scores, while Gradient Boosting lagged noticeably behind.

Table 3.1 Results obtained by XSS Fully featured utilization

	Precision		Recall		F1-Score		Accuracy
	XSS	Normal	XSS	Normal	XSS	Normal	
LR	0.9886	0.9309	0.9350	0.9487	0.9610	0.9397	0.9673
GB	0.6576	0.9654	0.9754	0.3876	0.7856	0.5531	0.7485
NB	0.6765	0.9543	0.8644	0.4367	0.7590	0.5992	0.8056
KM	0.8867	0.8733	0.8863	0.8754	0.8865	0.8744	0.9247
RF	0.8355	0.7865	0.9875	0.7986	0.9050	0.7925	0.9076
ANN	0.8966	0.9087	0.9845	0.7865	0.9385	0.8432	0.9898

SVM	0.9876	0.9765	0.9675	0.9876	0.9774	0.9820	0.8965
-----	--------	--------	--------	--------	--------	--------	--------

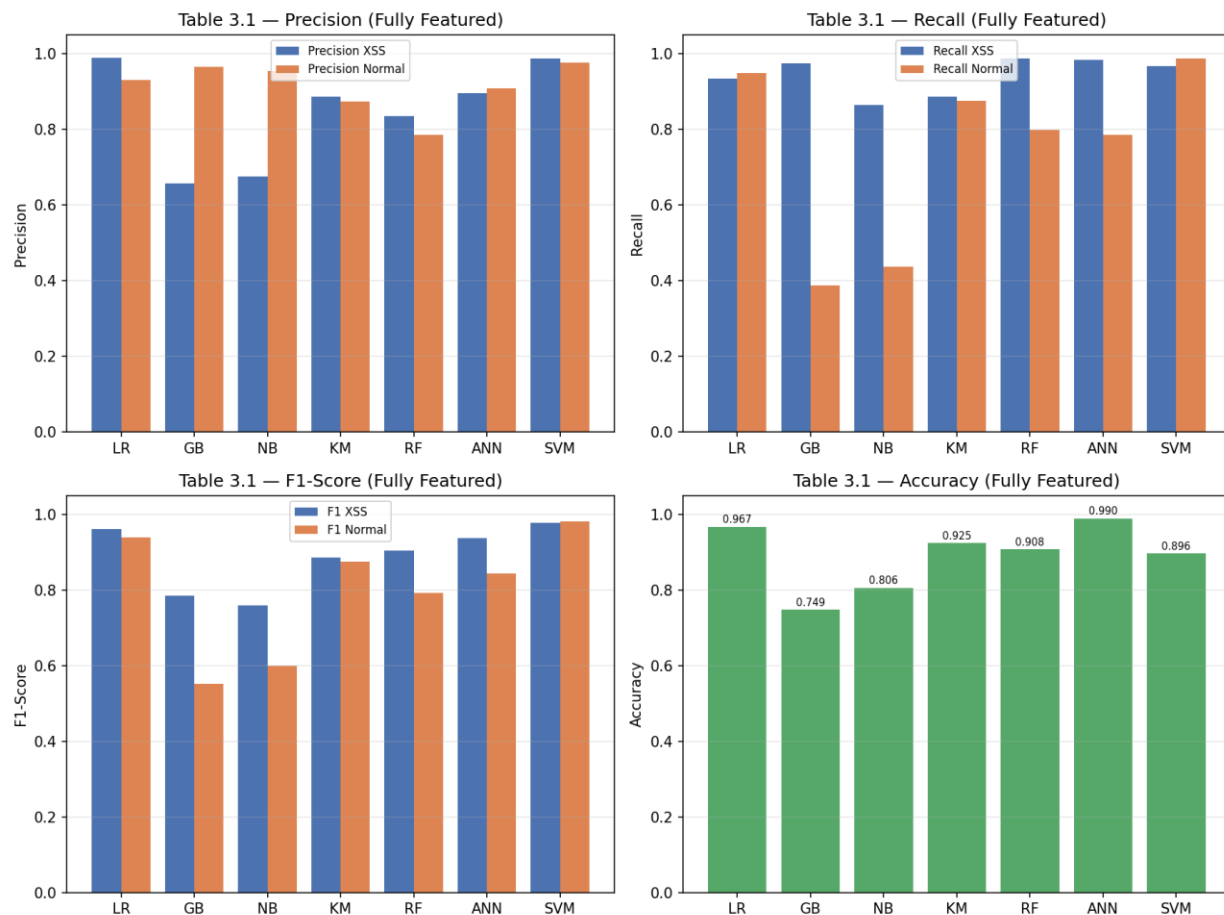


Figure 3.2 Precision, Recall, F1-Score and Accuracy by classifier (Table 3.1, Fully Featured Dataset)

3.8 Results obtained XSS attacks dataset utilization

Findings here line up with prior research in this space. The same seven algorithms LR, GB, NB, KM, RF, ANN and SVM were assessed, but this time trained on the reduced, preprocessed 77-feature set from Section 3.2.1 rather than the fully featured version behind Table 3.1; this difference in input features explains why the same classifiers post different scores across the two tables. Table 3.2 breaks down precision, recall, F1 and accuracy for both the attack and benign classes under this

configuration. Comparisons against other published datasets suggest the XSS Net benchmark holds an edge overall, but within our own results, ANN, SVM, and GB stand out as the strongest performers, each clearing 0.98 accuracy, while K-Means trails the field at 0.7638. Naive Bayes underperforms here as well, largely because its core assumption of feature independence doesn't hold for this dataset the 77 features are meaningfully interdependent, which undercuts the classifier's effectiveness.

Table 3.2 Results obtained by XSS attacks dataset set utilization

	Precision		Recall		F1-Score		Accuracy
	Attack	Benign	Attack	Benign	Attack	Benign	

LR	0.8898	0.8734	0.9834	0.7643	0.9343	0.8152	0.8974
GB	0.9776	0.6523	0.7876	0.8746	0.8724	0.7473	0.9870
NB	0.9853	0.9843	0.7643	0.7864	0.8608	0.8743	0.5644
KM	0.7645	0.7636	0.9874	0.7646	0.8618	0.7641	0.7638
RF	0.9873	0.8766	0.7467	0.7645	0.8503	0.8167	0.8765
ANN	0.8763	0.9874	0.7344	0.8747	0.7991	0.9276	0.9876
SVM	0.8764	0.9874	0.8743	0.8938	0.8753	0.9383	0.9874

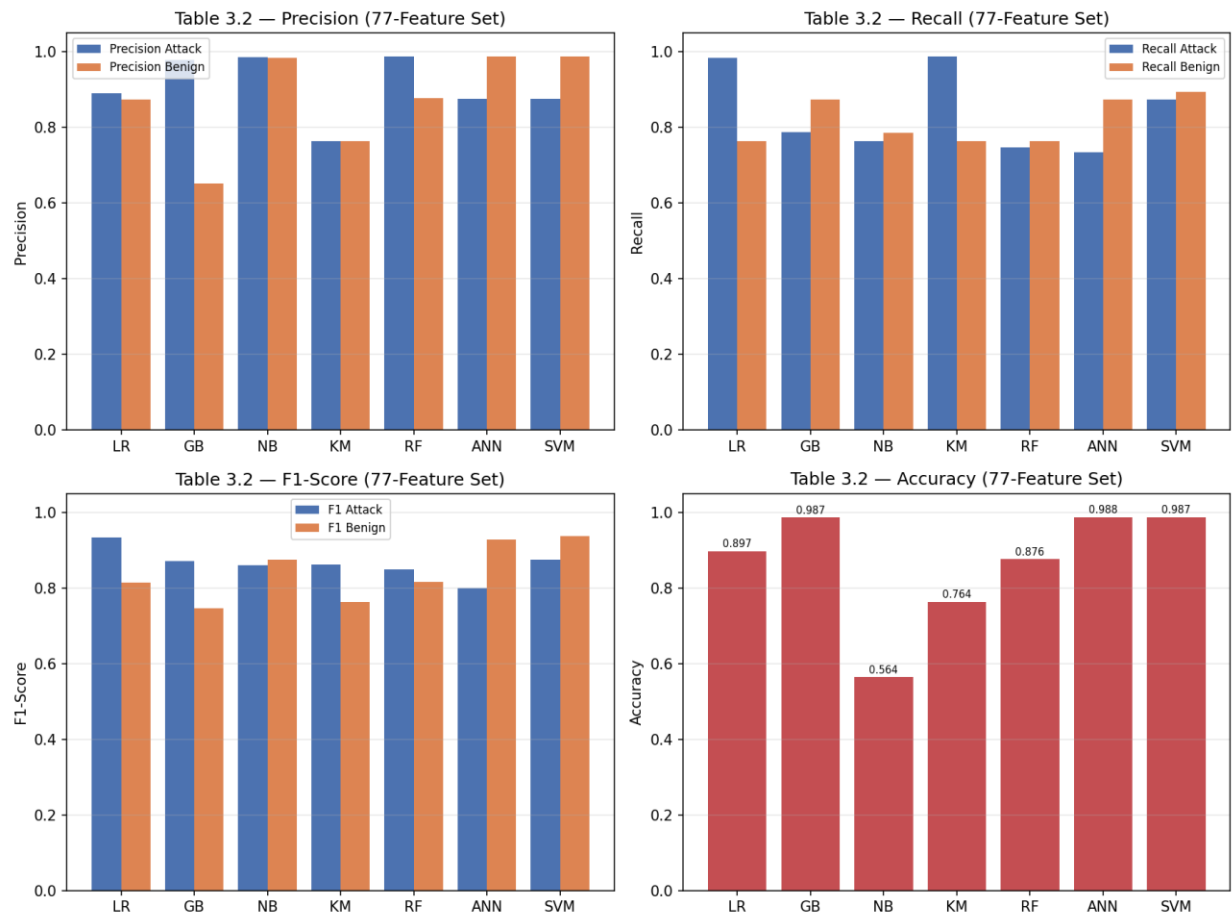


Figure 3.3 Precision, Recall, F1-Score and Accuracy by classifier (Table 3.2, 77-Feature Dataset)

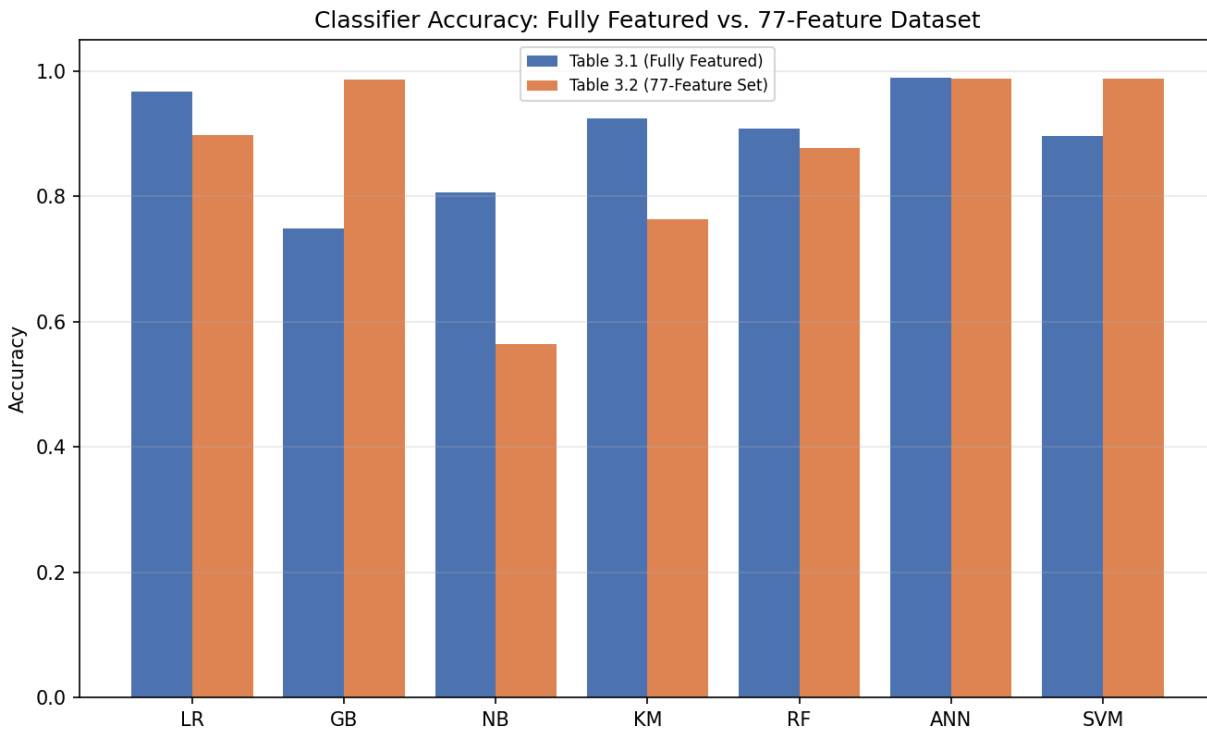


Figure 3.4 Accuracy comparison across classifiers: Fully Featured vs. 77-Feature Dataset

4. DISCUSSION

4.1 Quantitative techniques to analyze the classification algorithms

A few results from this study deserve closer examination. In broad terms, machine learning applies a numerical model to historical data in order to predict outcomes or perform classification and several such models and strategies were used in this work, described below.

4.1.1 Overview of Classification Algorithms Used

This study evaluated seven classification algorithms in total. Logistic Regression (LR) estimates class-membership probability through a logistic function, making it a reliable statistical baseline for binary classification (Cox, 1958). Gradient Boosting (GB) builds a strong classifier incrementally, adding successive weak learners that each correct the errors of the ensemble so far, guided by loss-function minimization (Natekin & Knoll, 2013). Naive Bayes (NB) classifies instances using posterior probability under the assumption

that features are mutually independent, an assumption that costs it accuracy when features are in fact correlated (Webb et al., 2010). K-Means (KM), adapted here with KPCA for anomaly-based classification, partitions data into K clusters by minimizing each point's distance to its cluster centroid (MacQueen, 1967). Random Forest (RF) is an ensemble technique that trains several decision trees and combines the results of all the trees to classify the data. RF can improve the accuracy of classification and prevent overfitting at the time of classification (Chopra et al., 2021). Artificial Neural Network (ANN) models mimic the structure of the brain with neurons and are very good at discovering complex non-linear relationships using hierarchical feature extraction. Support Vector Machine (SVM) is a technique which can find the optimum to separate hyperplane from high-dimensional feature space and it is useful for non-linear classification with no knowledge of the distribution of classes (Suthaharan, 2016). Consistent with recent comparative work, ensemble and deep-learning

methods such as RF and ANN outperformed the more traditional NB and LR classifiers on XSS-detection tasks in our results as well (Alhamyani & Alshammari, 2024; Kaur et al., 2023).

4.2 Proposed model evaluation metrics

To evaluate the performance of a classification model a few standard metrics are used throughout this study. They are calculated from a confusion matrix generated by applying the model to data that it has not been trained on and indicate for each class how many times the model was correct versus incorrect. Four counts lie at the heart of the matrix: the number of True Positives (TP) attack instances identified as attack; True Negatives (TN) normal instances identified as normal; False Positives (FP) normal instances identified as attack; and False Negatives (FN) attack instances identified as normal. The AU-ROC, which is especially suitable for measuring performance across multiple classes, illustrates the strength of a model's 0/1 prediction as the AUC value rises and is useful in visualizing the relationship between true-positive and false-positive rates across different thresholds. In other words, the AUC reflects how much of the IDS's alarms are actually "real alarm" and the smaller the false-positive rate, the less often the IDS screams "wolf".

4.3 Result based on XSS dataset utilization

ROC-AUC is used here as the main measure of model accuracy, reflecting the model's performance on positive and negative examples. The curve shows the trade-off between false positive and true positive rates and the area under the curve provides a summary of the overall performance of the classifiers, a high value (close to 1) indicates that the classifiers are able to discriminate well between the two classes, while a very low value (close to 0) indicates that the classifiers were not very effective.

4.4 Result based on RNN Classification

We applied the autoencoder-based classification approach to the SDN data set through an RNN implementation and obtained an ROC-AUC value of 0.984, indicating that the model

successfully classified positive and negative data about 98.4% of the time.

4.5 Result based on ROC with LSTM Classification

We performed this evaluation again with an LSTM-based classifier and measured its performance with the same SDN dataset and again assessed with the ROC-AUC. The model's accuracy in this case was an AUC of 0.982, a model which correctly identified positive and negative cases in approximately 98.2% of cases.

4.6 Discussion

This work gives a new perspective on anomaly detection by demonstrating the capabilities of machine-learning methods in this field. We implemented an RNN-Autoencoder and LSTM-Autoencoder based deep-learning pipeline that automatically learns the dimensionality reduction of the data without depending on features selected by hand. This type of deep learning is a promising way to identify network interference and can go beyond the limitations of conventional methods that rely on domain knowledge. Such a solution can be used in the application layer of an SDN controller in practice.

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusions

ANN achieved the best performance overall in the study, obtaining an accuracy of 98.98%, which demonstrates that deep-learning architectures are suitable for XSS detection. SVM and Logistic Regression also posted solid precision scores, while Gradient Boosting and Naive Bayes lagged behind the former likely hampered by class imbalance, the latter by feature interdependence violating its independence assumption.

Compared with alternative approaches, the method proposed here demands fewer server-side changes, which should ease adoption for web developers and since nothing changes on the end-user side, it can be rolled out with minimal user retraining. One trade-off is that it lets through some otherwise harmless HTML, which can interfere with certain page features. Because detection happens server-side, the approach can

only catch XSS attacks launched from the server it offers no protection against DOM-based XSS. We built the filter around a whitelist covering known XSS vulnerabilities across HTML5 and JavaScript, but a genuine novel, zero-day attack vector could still slip past it. Future work could extend this dataset with additional attributes to widen its coverage. While this study evaluated seven machine-learning algorithms, more advanced architectures transformer-based models, CNN-BiLSTM hybrids and ensemble-learning frameworks have shown stronger results in recent XSS-detection literature and are worth pursuing (Alhamyani & Alshammari, 2024). Since web archives also contain phishing and spam samples, a similar methodology could be applied to those attack types, extending this dataset's scope to a broader range of web-based threats.

5.2 Recommendations

Future work should test how well this method generalizes to other data sets and feature sets and we'd encourage experimenting with different learning-rate schedules and hyperparameter configurations. This study relied on binary classification to separate legitimate from malicious traffic, but distinguishing between individual attack types stored, reflected and DOM-based XSS specifically, would be a valuable next step, since each requires a somewhat different mitigation strategy. Later studies should move toward multi-class classification and adopt regularly refreshed training data to keep pace with XSS's constantly evolving tactics. This work lays groundwork for adaptable, deep-learning-based detection built on hybrid LSTM-autoencoder architectures capable of classifying multiple attack categories across both traditional and software-defined network environments (Alhamyani & Alshammari, 2024; Kaur et al., 2023).

6. REFERENCES

Aborujilah, A., & Musa, S. (2017). Cloud-based XSS HTTP attack detection using covariance matrix approach. *Journal of Computer Networks and Communications*, 2017.

- Alhamyani, R., & Alshammari, M. (2024). Machine Learning-Driven Detection of Cross-Site Scripting Attacks. *Information*, 15(7), 420. <https://doi.org/10.3390/info15070420>
- Barati, M., Abdullah, A., Udzir, N. I., Mahmood, R., & Mustapha, N. (2014, August). Distributed Denial of Service detection using hybrid machine learning technique. In *2014 International Symposium on Biometrics and Security Technologies (ISBAST)* (pp. 268-273). IEEE.
- Bhushan, K., & Gupta, B. B. (2018). Hypothesis test for low-rate XSS attack detection in cloud computing environment. *Procedia computer science*, 132, 947-955.
- Bremner-Barr, A., Brosh, E., & Sides, M. (2017, May). XSS attack on cloud auto-scaling mechanisms. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications* (pp. 1-9). IEEE.
- Chen, Y. W., Sheu, J. P., Kuo, Y. C., & Van Cuong, N. (2020, June). Design and implementation of IoT XSS attacks detection system based on machine learning. In *2020 European Conference on Networks and Communications (EuCNC)* (pp. 122-127). IEEE.
- Chen, Z., Jiang, F., Cheng, Y., Gu, X., Liu, W., & Peng, J. (2018, January). XGBoost classifier for XSS attack detection and analysis in SDN-based cloud. In *2018 IEEE international conference on big data and smart computing (bigcomp)* (pp. 251-256). IEEE.
- Chopra, A., Behal, S., & Sharma, V. (2021, March). Evaluating machine learning algorithms to detect and classify XSS attacks in IoT. In *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 517-521). IEEE.
- Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B*, 20(2), 215-232.

- Das, S., Mahfouz, A. M., Venugopal, D., & Shiva, S. (2019, July). XSS intrusion detection through machine learning ensemble. In 2019 IEEE 19th international conference on software Quality, Reliability and Security Companion (QRS-C) (pp. 471-477). IEEE.
- De la Hoz, E., De La Hoz, E., Ortiz, A., Ortega, J., & Prieto, B. (2015). PCA filtering and probabilistic SOM for network intrusion detection. *Neurocomputing*, 164, 71-81.
- Elsayed, M. S., Le-Khac, N. A., Dev, S., & Jurcut, A. D. (2020, August). XSSnet: A deep-learning model for detecting network attacks. In 2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks"(WoWMoM) (pp. 391-396). IEEE.
- Elsayed, M. S., Le-Khac, N. A., & Jurcut, A. D. (2020). InSDN: A novel SDN intrusion dataset. *IEEE Access*, 8, 165263-165284.
- Gu, Y., Li, K., Guo, Z., & Wang, Y. (2019). Semi-supervised K-means DDoS detection method using hybrid feature selection algorithm. *IEEE Access*, 7, 64351-64365.
- Gupta, B. B., & Badve, O. P. (2017). GARCH and ANN-based XSS detection and filtering in cloud computing environment. *International Journal of Embedded Systems*, 9(5), 391-400.
- Hamzah, K., Osman, M. Z., Anthony, T., Ismail, M., Abdullah, Z., & Alanda, A. (2024). Comparative Analysis of Machine Learning Algorithms for Cross-Site Scripting (XSS) Attack Detection. *JOIV: International Journal on Informatics Visualization*, 8(3-2), 1678-1685. <https://doi.org/10.62527/joiv.8.3-2.3451>
- Harish Kumar, J., & Godwin Ponsam, J. J. (2023). Cross Site Scripting (XSS) vulnerability detection using Machine Learning and Statistical Analysis. In 2023 International Conference on Computer Communication and Informatics (ICCCI) (pp. 1-9). IEEE. <https://doi.org/10.1109/iccci56745.2023.10128470>
- He, Z., Zhang, T., & Lee, R. B. (2017, June). Machine learning based XSS attack detection from source side in cloud. In 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud) (pp. 114-120). IEEE.
- Hijazi, A., El Safadi, A., & Flaus, J. M. (2018, December). A Deep Learning Approach for Intrusion Detection System in Industry Network. In *BDCSIntell* (pp. 55-62).
- Hou, J., Fu, P., Cao, Z., & Xu, A. (2018, October). Machine learning based XSS detection through NetFlow analysis. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)* (pp. 1-6). IEEE.
- Jiao, J., Ye, B., Zhao, Y., Stones, R. J., Wang, G., Liu, X., ... & Xie, G. (2017, September). Detecting TCP-based XSS attacks in Baidu cloud computing data centers. In 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS) (pp. 256-258). IEEE.
- Karan, B. V., Narayan, D. G., & Hiremath, P. S. (2018, December). Detection of XSS attacks in software defined networks. In 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS) (pp. 265-270). IEEE.
- Kasim, Ö. (2020). An efficient and robust deep learning based network anomaly detection against distributed denial of service attacks. *Computer Networks*, 180, 107390.

- Kaur, J., Garg, U., & Bathla, G. (2023). Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review. *Artificial Intelligence Review*, 56(11), 12725-12769.
<https://doi.org/10.1007/s10462-023-10433-3>
- Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1), 1-22.
- Khuphiran, P., Leelaprute, P., Uthayopas, P., Ichikawa, K., & Watanakeesuntorn, W. (2018, November). Performance comparison of machine learning models for XSS attacks detection. In *2018 22nd International Computer Science and Engineering Conference (ICSEC)* (pp. 1-4). IEEE.
- Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2014). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1), 14-76.
- Li, Q., Meng, L., Zhang, Y., & Yan, J. (2018, September). XSS attacks detection using machine learning algorithms. In *International Forum on Digital TV and Wireless Multimedia Communications* (pp. 205-216). Springer, Singapore.
- MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* (Vol. 1, pp. 281-297). University of California Press.
- Mohammed, S. S., Hussain, R., Senko, O., Bimaganbetov, B., Lee, J., Hussain, F., ... & Bhuiyan, M. Z. A. (2018, October). A new machine learning-based collaborative XSS mitigation mechanism in software-defined network. In *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)* (pp. 1-8). IEEE.
- Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7, 21.
- OWASP. (2021). OWASP Top 10:2021 - A03:2021 Injection. Open Web Application Security Project.
<https://owasp.org/Top10/2021/>
- Pandey, V. C., Peddoju, S. K., & Deshpande, P. S. (2018). A statistical and distributed packet filter against XSS attacks in Cloud environment. *Sādhana*, 43(3), 1-9.
- Priya, S. S., Sivaram, M., Yuvaraj, D., & Jayanthiladevi, A. (2020, March). Machine learning based XSS detection. In *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)* (pp. 234-237). IEEE.
- Rahman, O., Quraishi, M. A. G., & Lung, C. H. (2019, July). XSS attacks detection and mitigation in SDN using machine learning. In *2019 IEEE world congress on services (SERVICES)* (Vol. 2642, pp. 184-189). IEEE.
- Rahul, R. K., Anjali, T., Menon, V. K., & Soman, K. P. (2017, September). Deep learning for network flow analysis and malware classification. In *International Symposium on Security in Computing and Communication* (pp. 226-235). Springer, Singapore.
- Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). A deep learning approach to network intrusion detection. *IEEE transactions on emerging topics in computational intelligence*, 2(1), 41-50.
- Singh, R., Kumar, H., & Singla, R. K. (2015). An intrusion detection system using network traffic profiling and online sequential extreme learning machine. *Expert Systems with Applications*, 42(22), 8609-8624.
- Sundaram, A. (1996). An introduction to intrusion detection. *Crossroads*, 2(4), 3-7.
- Suthaharan, S. (2016). Support vector machine. In *Machine learning models and algorithms for big data classification* (pp. 207-235). Springer, Boston, MA.

- Vinayakumar, R., Soman, K. P., Poornachandran, P., & Sachin Kumar, S. (2018). Evaluating deep learning approaches to characterize and classify the DGAs at scale. *Journal of Intelligent & Fuzzy Systems*, 34(3), 1265-1276.
- Wahab, O. A., Bentahar, J., Otrouk, H., & Mourad, A. (2017). Optimal load distribution for the detection of VM-based XSS attacks in the cloud. *IEEE transactions on services computing*, 13(1), 114-129.
- Webb, G. I., Keogh, E., & Miikkulainen, R. (2010). Naïve Bayes. *Encyclopedia of machine learning*, 15, 713-714.
- Yang, C. (2019). Anomaly network traffic detection algorithm based on information entropy measurement under the cloud computing environment. *Cluster Computing*, 22(4), 8309-8317.
- Zekri, M., El Kafhali, S., Aboutabit, N., & Saadi, Y. (2017, October). XSS attack detection using machine learning techniques in cloud computing environments. In *2017 3rd international conference of cloud computing technologies and applications (CloudTech)* (pp. 1-7). IEEE.

