Federated Learning for XSS Detection: A Privacy-Preserving Approach

Mahran Jazi[a] and Irad Ben-Gal[b]
Department of Industrial Engineering, Tel Aviv University, Tel Aviv, Israel

Keywords: Federated Learning, Cross-Site Scripting (XSS) Detection, On-Device Learning, Non-IID Data Distribution, Threat Detection in Web Applications.

Abstract: Collaboration between edge devices has increased the scale of machine learning (ML), which can be attributed to increased access to large volumes of data. Nevertheless, traditional ML models face significant hurdles in securing sensitive information due to rising concerns about data privacy. As a result, federated learning (FL) has emerged as another way to enable devices to learn from each other without exposing user's data. This paper suggests that FL can be used as a validation mechanism for finding and blocking malicious attacks such as cross-site scripting (XSS). Our contribution lies in demonstrating the practical effectiveness of this approach on a real-world dataset, the details of which are expounded upon herein. Moreover, we conduct comparative performance analysis, pitting our FL approach against traditional centralized parametric ML methods, such as logistic regression (LR), deep neural networks (DNNs), support vector machines (SVMs), and k-nearest neighbors (KNN), thus shedding light on its potential advantages. The dataset employed in our experiments mirrors real-world conditions, facilitating a meaningful assessment of the viability of our approach. Our empirical evaluations reveal that the FL approach not only achieves performance on par with that of centralized ML models but also provides a crucial advantage in terms of preserving the privacy of sensitive data.

1 INTRODUCTION

Today's digital landscape is crowded with edge devices that are multiplying at an alarming rate. As a result, an unimaginably large amount of personal data, complete with different aspects of users' lives, such as multimedia content and text information, is being accumulated. Using this private data to support machine learning (ML) in user applications has become more common. However, the conventional approach of centralizing the ML training process on powerful servers presents a problem. While data originates and applications execute on edge devices, centralized servers amass substantial portions of user data, triggering significant privacy concerns (McMahan et al., 2017).

This centralization creates a fundamental conflict: on the one hand, centralized servers participate in supplying the computational searching ability and storage for training complicated multilayer models; on the other hand, there exist some safety risks for users. The downside of centralization is possible data security threats and abuse of users' data, as well as large communication costs. As a result, available solutions for on-client machine learning have had to be efficient and privacy-preserving to avoid sending data to large repositories. Therefore, recently, the idea of federated learning (FL) has emerged as a possible solution to both concerns (McMahan et al., 2017). FL implements a distributed collaborative learning algorithm, which does not necessitate storing user data in the cloud or on a central server, as illustrated in 1. Under this setup, each client keeps its local private training dataset and never relinquishes control over sensitive information. Instead of transmitting raw data, clients

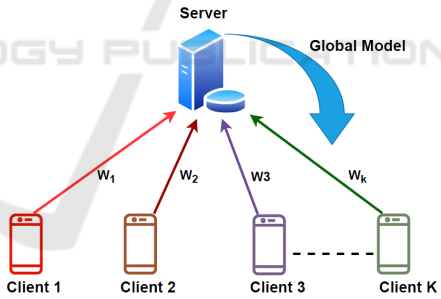

Figure 1: Federated learning scheme.

[a] https://orcid.org/0000-0001-6432-3800
[b] https://orcid.org/0000-0003-2411-5518

Jazi, M. and Ben-Gal, I.
Federated Learning for XSS Detection: A Privacy-Preserving Approach.
DOI: 10.5220/0012921800003838

In *Proceedings of the 16th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2024) - Volume 1: KDIR*, pages 283-293
ISBN: 978-989-758-716-0; ISSN: 2184-3228

only share their local model parameters, like neural network weights, with central servers. These models are often structured similarly and undergo aggregation averaging before being re-distributed to clients.

In addition, nowadays, the world increasingly operates with Internet-based services and web applications (Kotzur, 2022). This overreliance on web-based communication leads to increased website attacks that seek to breach a system's vulnerabilities and corrupt the devices and data, destroying them. One particular threat to note is Cross-Site Scripting (XSS), which remains one of the Open Web Application Security Projects (OWASP) (OWASP, 2017) and has been identified by past studies as a highly persistent issue. Conventional mechanisms for patching security holes in web applications rely on a database of information on known attack signatures(Ariu and Giacinto, 2011). Nevertheless, XSS attacks usually take advantage of vulnerabilities in user input specification or leverage the space between client-side and server-side defenses (Rocha and Souto, 2014; Lee et al., 2022). FL could mitigate web-based communication safety issues in the same way it enhances privacy by decentralized data. Furthermore, novel strategies are needed to mitigate these security challenges by empowering ML. FL can increase the resistance of web applications against XSS attacks, leveraging its principles to preserve user privacy and enhance security.

1.1 Contributions

We propose a novel system that employs FL to detect XSS attacks, addressing the limitations of traditional methods. This innovative approach allows users to leverage shared models while preserving decentralized storage's privacy and scalability benefits. In practical tests, where we employed various ML models within the FL-based system, we effectively demonstrated its capacity to detect and counter XSS attacks. Through a series of experiments, we thoroughly evaluated the effectiveness of diverse ML algorithms in detecting XSS attacks using accuracy metrics. Our assessment included traditional logistic regression (LR) and deep neural network (DNN) algorithms as centralized models, allowing us to juxtapose their performance with that of FL. By comparing the performances of traditional LR and DNN algorithms with that of FL, our goal was to determine the most effective and efficient approach for accurately detecting attacks. Notably, the privacy-preserving nature of FL ensures that clients do not share any private data, addressing the key concerns associated with collaborative learning. Our results are based on the iid and non-iid data distribution settings.

2 BACKGROUND

2.1 Web Applications and JavaScript

Web applications refer to software programs designed to perform specific tasks and are typically requested by a client's web browser over the internet (Ndegwa, 2016). These applications are hosted on remote servers and accessed through web browsers. These tools include two main components: server-side scripts, such as Java Servlets, ASP, and PHP, which manage the processing and retrieval of data from the backend database; and client-side scripts, such as HTML and Java Applets, which are responsible for presenting the information to a user in their web browser.

Examples of typical web applications include email services and e-commerce platforms, which may require server-side processing, as well as applications that do not require any processing on the server. To handle HTTP requests from a client, a web server is necessary, an application server is needed to execute the requested tasks, and a database is used to store information when necessary.

Various security implications arise from poor programming of the web applications (Meyer and Cid, 2008). These bugs can be exploited to gain unauthorized access to a server and its associated databases or steal sensitive data, like credit card information. The term "web application attacks" is used for these types of attacks on web applications.

JavaScript is an object-oriented scripting language commonly employed in designing and implementing dynamic websites. It does not involve server-side processing like other programming languages but relies purely on a client browser to execute the source code. However, JavaScript can also be misused by hackers who want to distribute malicious scripts through different means. For example, they perform Cross-Site Scripting (XSS), Passive Downloads, or SQL injection attacks (Wei-Hong et al., 2013).

2.2 Cross-Site Scripting Attacks

Cross-site scripting (XSS) has become a prevalent way of attacking many websites (Lee et al., 2022). OWASP categorizes such attacks among the top ten in terms of how incapacitating they can be. The purpose of an XSS attack is to place destructive content within a valid web page or application and execute it on the victim's browser. This occurs when a blameless user visits a webpage or web app that includes damaging programming, which then runs on his/her web browser, allowing the attacker to gain unautho-

rized entry into private information, including cookies/user profiles or installing malware. Commonly used conduits for XSS are message boards, forums, and web pages where users leave comments.

Reflected XSS attacks, stored XSS attacks, and DOM-based XSS attacks are the three major types of these attacks (Galán et al., 2010). The methods each uses to inject attacker codes into applications differ concerning these three attack types and the means through which they affect code execution. Most authors exclude DOM-based XSS attacks when categorizing XSS attacks because these attacks are vulnerable to the script used by web browsers, unlike Reflected and Stored XSS attacks, which exploit vulnerabilities in web applications (Klein, 2005). Reflected and Stored XSS attacks involve injecting script code through an HTTP request. Reflected XSS attacks, also known as nonpersistent XSS attacks, are commonly used to steal sensitive data, such as cookies, by executing malicious scripts on the victims' machines. This type of attack is executed immediately in the victim's browser since the script is included in the HTTP response.

On the other hand, stored XSS attacks are more dangerous because they can affect multiple users who access the infected page. This attack entails directly injecting the script or payload into the target page's database, which allows the attacker to keep running their malicious code.

The third kind of XSS attack is DOM-based XSS, and it is unique in that it does not rely on a vulnerability inherent in a web application itself. Rather, these attacks exploit vulnerabilities within the document object model (DOM) of a web browser to inject malicious code inside targeted pages. The attacker can do this by engineering a malevolent URL that, when clicked, will insert the code straight into the DOM of that page.

Although all kinds of XSS attacks may have different features and execution methods, they all aim to exploit web applications to execute software scripts. Consequently, developers and website administrators must be familiar with this class of attacks and protect their systems against them.

2.3 Data Privacy in XSS Detection and the Role of Federated Learning

A critical aspect of XSS detection involves analyzing the data or scripts embedded in web pages. While the scripts might not always contain sensitive information, they are often associated with user-specific contexts, such as session data, browsing history, or user interactions with web applications. This context can reveal users' private information, such as their browsing habits, preferences, and even personal identifiers.

Traditionally, XSS detection models have relied on centralized servers to aggregate and analyze this data, raising significant privacy concerns. Centralized storage and processing of such data can expose it to potential breaches, misuse, or unauthorized access, compromising user privacy. This is particularly concerning in scenarios involving edge devices, where data is generated and consumed locally, such as in IoT environments.

Federated Learning (FL) presents an innovative solution to this privacy challenge. By allowing devices to collaboratively train models without sharing raw data with a central server, FL ensures that sensitive information remains on the user's device. In XSS detection, each device can contribute to improving the detection model by sharing only model updates rather than the underlying data.

This approach is particularly valuable for protecting data privacy in edge computing environments, where data decentralization is both a necessity and a strength. By applying FL to XSS detection, we can enhance the security and privacy of web applications while still leveraging the collective intelligence of distributed devices.

These privacy considerations motivate the choice of FL for XSS detection. Our work explores this novel application of FL to XSS detection, providing a framework for maintaining high detection accuracy without compromising user privacy. Through comparative analysis with traditional centralized machine learning models, we demonstrate the effectiveness of FL in this context, highlighting its potential to revolutionize the web security field.

3 RELATED WORK

3.1 Cross-Site Scripting Detection

In 2009, Likrash et al. worked on predicting malicious JavaScript code using multiple ML classifiers. These classifiers were used to determine which features of the JavaScript code could help their model determine potentially malicious code (Likarish et al., 2009). The classifiers used in this study were naïve Bayes, ADTree , support vector machine (SVM), and RIPPER classifiers. The authors of this study used a 10-fold cross-validation technique to train and test the models; thus, the data were divided into ten segments: 9 for training and one for testing. However, this process was performed ten times, so each segment was utilized in the training and testing phases at least once

(Likarish et al., 2009). In their work, the authors achieved a precision value ((number of correctly labeled malicious scripts)/(total number of scripts that are marked as malicious)) of 0.92% and a recall rate ((number of correctly labeled malicious scripts)/(total number of malicious scripts)) of 0.787% (Likarish et al., 2009). One concern with this study was that the training set contained 50,000 benign codes and only 62 malicious codes without oversampling the malicious code, which might explain the low recall value.

Komiya et al. (Komiya et al., 2011) used ML techniques, such as SQL injection and XSS attacks, adapted to changes in code characteristics to predict malicious web code. The first stage was the learning process. In this stage, the classifier extracted features from malicious or nonmalicious web code from each training dataset using a feature vector. The vector contained the weight of each feature (term), which was the number of occurrences calculated using the term frequency-inverse document frequency (TF-IDF) method. The second process was the classification process, which used the criteria constructed from the learning process to classify the user input. The authors constructed two separate classifiers, one for XSS attacks and another for SQLIAs(Komiya et al., 2011). The utilized classifiers included an SVM (with a linear kernel), another SVM (with a polynomial kernel), a third SVM (with a Gaussian kernel), naive Bayes, and K-nearest neighbors (KNN). The KNN classifier yielded the highest precision (0.991), and the SVM with a Gaussian kernel yielded the highest accuracy (99.16%). The principal concern regarding this study was that the dataset used for training and testing was relatively small and might not have reflected real-world web attacks(Komiya et al., 2011). Another experiment conducted by Nunan involved XSS attacks (Nunan et al., 2012). By depending on web document content and URLs, the authors aimed to detect malicious pages using ML techniques. Different classification algorithms were used to extract features that helped predict XSS attacks. In this experiment, the authors focused on detecting web page obfuscation by encoding hexadecimal, decimal, octal, Unicode, Base64, and HTML reference characters. The employed classification algorithms included naive Bayes and SVM classifiers. They also performed this experiment using 10-fold cross-validation (Nunan et al., 2012). Wei-Hong et al. worked on detecting malicious scripts by using static analysis techniques to extract features and an SVM to classify scripts (Wei-Hong et al., 2013). The authors extracted features first based on previous work performed by other researchers and second by manually analyzing the data. In their work, utilizing an SVM, they reached an accuracy of 96.59% on the training set and an accuracy of 94.38% on the testing set (Wei-Hong et al., 2013). Other researchers have used ML techniques to distinguish between obfuscated and nonobfuscated scripts (Aebersold et al., 2016). To reach this goal, they used the following classifiers while depending on Azure ML: average perceptron (AP), Bayes point machine (BPM), boosted decision tree (BDT), decision forest (DF), decision jungle (DJ), locally deep SVM (LDSVM), LR, neural network (NN), and SVM classifiers. The authors studied the ability of these classifiers to detect malicious scripts; however, no malicious scripts were included in the training dataset that was used to build the models. The BDT classifier achieved the highest precision (100%), with a recall of 47.71% (Aebersold et al., 2016). Mereani et al. (Mereani and Howe, 2018) aimed to build classifiers to predict a persistent (on-storage) XSS attack in a Java script using ML techniques. Persistent XSS attacks occur when a hacker injects his or her code and saves it in the database of the target web application. Whenever the web application is accessed, the script runs on the user's browser. The authors used three classifiers in their work: an SVM, a KNN, and a random forest. The conventional approach to XSS detection typically involves extracting certain features based on experience and subsequently determining if it constitutes an XSS attack using rule-based matching methods. However, this methodology struggles to identify increasingly intricate XSS attack patterns. With the swift advancements in machine learning, an expanding cohort of researchers has endeavored to address network security issues through machine learning algorithms, with particular emphasis on XSS attack detection, resulting in notable advancements (Yan et al., 2022; Wu et al., 2021a; Wu et al., 2021b; Wu et al., 2021c; Wu et al., 2019). (ZHOU et al., 2019) proposed a model that combines a multilayer perceptron with a hidden Markov model (HMM). (Luo et al., 2020) developed a URL feature representation method by analyzing existing URL attack detection technologies and put forward a multi-source fusion method based on a deep learning model. This approach enhances the overall accuracy and system stability of the XSS detection system.

3.2 Federated Learning

The inception of FL traces its origins to 2017, marked by the unveiling of this innovative paradigm by Google in their seminal paper (McMahan et al., 2017). This pioneering endeavor introduced a decentralized approach, denoted FL, meticulously designed to preserve the privacy of the data belonging to participating clients. In the FL domain, a central server or aggregator assumes a central role in or-

chestrating a consortium of clients, enabling collaborative data analysis through a shared model. Crucially, FL upholds the principle of data sovereignty, ensuring that each client maintains absolute control over their data, which remains confined within the boundaries of their devices. Within this framework, the model is regarded as communal property shared among the clients, with the sole exchanges encompassing parameter updates. Additionally,(McMahan et al., 2017) proposed a novel decentralized learning technique termed federated averaging that models the performance of a convolutional neural network (CNN) using diverse types of data such as the MNIST, CIFAR-10, and LSTM datasets. Numerous studies have extended the scope of FL. For instance, they may focus on the challenges associated with system heterogeneity and seek to minimize the incurred communication overhead. In their work, Bonawitz et al. (Bonawitz et al., 2017) implemented a secure and efficient FL algorithm with a fixed number of rounds, ensuring low communication overhead and high robustness, especially when handling high-dimensional data received from clients. Addressing uplink costs, Konevcny et al. (Konečnỳ et al., 2016) introduced methods based on structured and sketched updates, showcasing significant communication overhead reductions (by two orders of magnitude).To help with training, they employed techniques such as correcting the momentum and clipping the local gradient to significantly reduce the communication bandwidth overhead in deep gradient compression (DGC) (Lin et al., 2017). Additionally, Hsieh et al. (Hsieh et al., 2020), Li et al. (Li et al., 2020b), and Shamir et al. (Shamir et al., 2014) developed novel distributed learning algorithms by employing multiple minibatches and full-batch stochastic gradient descent (SGD) to alleviate communication overheads and improve overall efficiency of FL. This is believed to be the first time that FL has been used for user privacy protective XSS attack detection (McMahan et al., 2017; Yang et al., 2019; Li et al., 2020a; Zhang et al., 2021; Kairouz et al., 2021).

4 PROBLEM FORMULATION

This section describes our proposed FL scheme, which runs FL on a set of clients. We consider N clients engaged in a classification task, where the goal is to learn a function that maps every input data point to the correct class out of K possible options. Each client n has access to its own private data $\mathcal{D}_n = \{x_i^n\}_{i=1}^{M_n}$ consisting of M_n inputs and their corresponding labels y_i. All the labels are hard-decision vectors formed over the set of all classes. Each client n has a model (e.g., a DNN) with p parameters (e.g., weights): $\omega^n \in \mathbb{R}^p$. We follow conventional FL and assume that all clients have the same architecture for their models so they can be easily averaged.

Let $l(\omega, x, y)$. be the loss incurred on a training data point (x, y). The local training loss function of client n is then

$$\mathcal{L}_n(\omega^n) \triangleq \sum_{x \in \mathcal{D}_n} l(\omega^n, x, y(x)). \tag{1}$$

The goal of the training process in FL is to learn a common model ω that minimizes the total loss induced across all clients, which is defined as follows:

$$\mathcal{L}(\omega) = \sum_{n=1}^{N} \mathcal{L}_n(\omega). \tag{2}$$

The iterations t of the employed FL algorithm consist of two parts. First, the server collects the client models and computes the average model:

$$\overline{\omega}(t) = \frac{1}{N} \sum_{n=1}^{N} \omega^n(t). \tag{3}$$

Then, each client performs a local SGD step to update the average model, with a momentum parameter $0 \leq \beta < 1$:

$$\omega^n(t+1) = \overline{\omega}(t)\eta(t)v(t+1) \tag{4}$$

where $\eta(t)$ is the step size sequence and

$$v(t+1) = \beta v(t) + g_n(\overline{\omega}(t)) \tag{5}$$

Which coincides with the standard SGD strategy for $\beta = 0$. The stochastic gradient $g_n(\overline{\omega}(t))$ is obtained concerning a random subset of data points $\mathcal{S}_n \subset \mathcal{D}_n$ of size B (i.e., the batch size):

$$g_n(\overline{\omega}(t)) = \sum_{x \in \mathcal{S}_n} \nabla l(\overline{\omega}(t), x, y(x)). \tag{6}$$

5 EXPERIMENTAL RESULTS

The experiments were conducted using Google Colab, running the datasets and models within the Python environment. We report the percentage of accurately classified data points in the test dataset ("accuracy") for the federated learning (FL) model obtained after training.

5.1 Datasets

5.1.1 XSS Dataset

We utilized a balanced XSS dataset comprising scripts from multiple sources. Two datasets containing malicious and benign JavaScript programs were

Table 1: Datasets containing malicious and benign scripts.

Dataset Type	Dataset Source
Malicious Scripts	(Mereani and Howe, 2018)
Benign Scripts	(Mereani and Howe, 2018)

gathered, with the sources listed in Table 1. The dataset consists of 62 attributes and 24,097 data instances.

Selecting the features to be trained by FL models is challenging due to the vast number of available design options. Feature selection is typically divided into two categories.

- **Structural Features.** This refers to the complete set of non-alphanumeric characters, including five additional combinations. For example, a hacker may add unnecessary commands (¡!) or spaces between lines.
- **Behavioral Features.** These are specific commands or functions that may be used in a malicious JavaScript, such as the `(Var)` function. Table 2 outlines both the structural and behavioral features utilized in our experiments.

5.1.2 CICIDS2017 Dataset

The CICIDS2017 dataset (Sharafaldin et al., 2018), created by the Canadian Institute for Cybersecurity (CIC), offers a comprehensive collection of network traffic data representing various cyber threats and normal network behavior. The dataset consists of 85 features with 458,968 data instances. Preprocessing steps included checking for null values, converting categorical objects to numerical values, and normalizing the data to eliminate outliers.

5.2 Model Architectures and Training

We implemented and tested four machine-learning models:

- **Logistic Regression (LR):** A basic linear model used for binary classification tasks. This model served as a baseline for our comparisons.
- **Multilayer Perceptron (MLP or 2NN):** A neural network with two hidden layers containing 200 units and ReLU activation functions as in (McMahan et al., 2017). The architecture is simple yet effective for binary classification tasks involving malicious and benign labels.
- **Support Vector Machine (SVM):** A powerful model used for binary classification, particularly effective in high-dimensional spaces. We used a Radial Basis Function (RBF) kernel, which is a common choice for non-linear classification. The RBF kernel maps the input data into a higher-dimensional space, which makes it easier to classify using a linear decision boundary.
- **k-Nearest Neighbors (KNN):** A simple yet effective non-parametric method for classification tasks. We set k=5 as a default value, balancing bias and variance.

All models were trained using Stochastic Gradient Descent (SGD) where applicable (for LR and MLP) with the following parameters:

- **Learning rate:** $\eta = 0.01$
- **Momentum:** $\beta = 0.9$
- **Batch size:** $B = 32$
- **Local epochs:** $E = 1$ (i.e., SGD operated over the local dataset once)
- **Communication rounds:** 100

The KNN model's classification is non-iterative, so the parameters related to SGD do not apply. Instead, the model computes distances between data points and assigns labels based on the majority class among the nearest neighbors.

Each experiment was run ten times, and the results presented are averages across these runs, with error bars representing one standard deviation.

To simulate a realistic Federated Learning (FL) environment, we divided the XSS and CICIDS2017 datasets into several shards, assigning each shard to a different client and ensuring each client had its private data. We followed an 80:20 train-test split, reserving 20% of the data for evaluating the model's performance on unseen data.

5.3 Reproducibility and Code Availability

We implemented all models using standard libraries in Python. To ensure reproducibility, the models' detailed architectures, preprocessing steps, and training configurations are provided. The code, including data preprocessing scripts, model architectures, and the FL implementation, will be made available in a public repository upon the paper's acceptance. The datasets used are publicly accessible and cited accordingly.

5.4 IID Data Distribution

The results obtained for independent and identically distributed (IID) data are presented in Table 3. In this setting, ten clients were utilized, each with random data points from the specific datasets (XSS and CICIDS 2017); thus, the data distributions among the

Table 2: Behavioral and structural features (Mereani and Howe, 2018).

Features	Description	Type
Readability	The number of alphabetical characters.	Behavioral
Objects	Document, window, I frame, location.	Behavioral
Events	Onload, Onerror.	Behavioral
Methods	createElement, String.fromCharCode.	Behavioral
Tags	DIV, IMG, <script>.	Behavioral
Attributes	SRC, Href, Cookie.	Behavioral
Reserve	Var.	Behavioral
Functions	eval().	Behavioral
Protocol	HTTP.	Behavioral
External File	.js file.	Behavioral
Punctuation	<, #, $, @.	Structural
Combinations	”¿¡”, ”==”.	Structural

Table 3: Performance of FL and Traditional Centralized models in binary classification on XSS and CICIDS2017 datasets based on IID data distribution.

Dataset	Model	Accuracy	Precision	Recall	F1 Score
XSS	FL model (LR)	98.9%	99.9%	97.3%	98.6%
	Traditional Centralized model (LR)	99.9%	99.9%	99.8%	99.9%
	FL model (DNN)	99.9%	99.9%	99.9%	99.9%
	Traditional Centralized model (DNN)	99.9%	99.9%	99.8%	99.9%
	FL model (SVM)	99.9%	99.9%	99.9%	99.9%
	Traditional Centralized model (SVM)	99.96%	99.96%	99.96%	99.96%
	FL model (KNN)	99.7%	99.7%	99.7%	99.7%
	Traditional Centralized model (KNN)	99.90%	99.90%	99.90%	99.90%
CICIDS2017	FL model (LR)	94.01%	89.36%	90.0%	89.86%
	Traditional Centralized model (LR)	97.9%	94.72%	98.33%	96.49%
	FL model (DNN)	98.48%	96.49%	98.33%	97.40%
	Traditional Centralized model (DNN)	99.8%	99.3%	99.9%	99.6%
	FL model (SVM)	98.41%	96.48%	98.09%	97.28%
	Traditional Centralized model (SVM)	99.31%	99.04%	98.57%	98.80%
	FL model (KNN)	96.49%	90.54%	98.09%	94.17%
	Traditional Centralized model (KNN)	99.38%	98.12%	99.76%	98.93%

clients are similar. The experiment aims to identify anomalies in the data. Our results, as an example in Figure 2, confirmed that FL could reach a comparable performance level to traditional centralized models after a few communication rounds while maintaining data privacy and never sharing any sensitive data with the central server. Furthermore, the results for the binary classification of XSS and CICIDS2017 datasets, presented in Table 3, include the performance of additional classifiers, such as SVM and KNN, along with LR and DNN models. These results demonstrate that federated learning offers superior accuracy, precision, recall, and F1 score across multiple models addressing the XSS issue.

5.5 Non-IID Data Distribution

The results for non-independent and identically distributed (non-IID) data are presented in Table 4. In this setting, each client i had data points belonging to class i of the XSS and CICIDS2017 datasets, with $i = 1, 2$. To create the most non-IID case, client one was assigned all the malicious data points, while client two was assigned all the benign data points. The results show that the federated model offers slightly lower scores in some cases than the centralized model. However, the difference is not substantial, indicating that the horizontal FL system can perform well in addressing the XSS problem using FedAvg as the aggregating algorithm. Figure 3 provides an example of the behavior of the Logistic Regression (LR) and Deep Neural Network (DNN) mod-

Table 4: Performance of FL and Traditional Centralized models in binary classification on XSS and CICIDS2017 datasets based on non-IID data distribution.

Dataset	Model	Accuracy	Precision	Recall	F1 Score
XSS	FL model (LR)	98.77%	99.9%	97.08%	98.51%
	Traditional Centralized model (LR)	99.9%	99.9%	99.8%	99.9%
	FL model (DNN)	99.85%	99.9%	99.65%	99.82%
	Traditional Centralized model (DNN)	99.9%	99.9%	99.8%	99.9%
	FL model (SVM)	99.91%	99.90%	99.90%	99.90%
	Traditional Centralized model (SVM)	99.96%	99.96%	99.96%	99.96%
	FL model (KNN)	99.81%	99.95%	99.60%	99.77%
	Traditional Centralized model (KNN)	99.90%	99.90%	99.90%	99.90%
CICIDS2017	FL model (LR)	95.8%	89.7%	89.8%	93.1%
	Traditional Centralized model (LR)	97.9%	94.72%	98.33%	96.49%
	FL model (DNN)	96.08%	97.14%	89.04%	92.91%
	Traditional Centralized model (DNN)	99.8%	99.3%	99.9%	99.6%
	FL model (SVM)	97.73%	97.79%	97.73%	97.74%
	Traditional Centralized model (SVM)	99.31%	99.04%	98.57%	98.80%
	FL model (KNN)	96.63%	89.97%	98.33%	93.97%
	Traditional Centralized model (KNN)	99.38%	98.12%	99.76%	98.93%

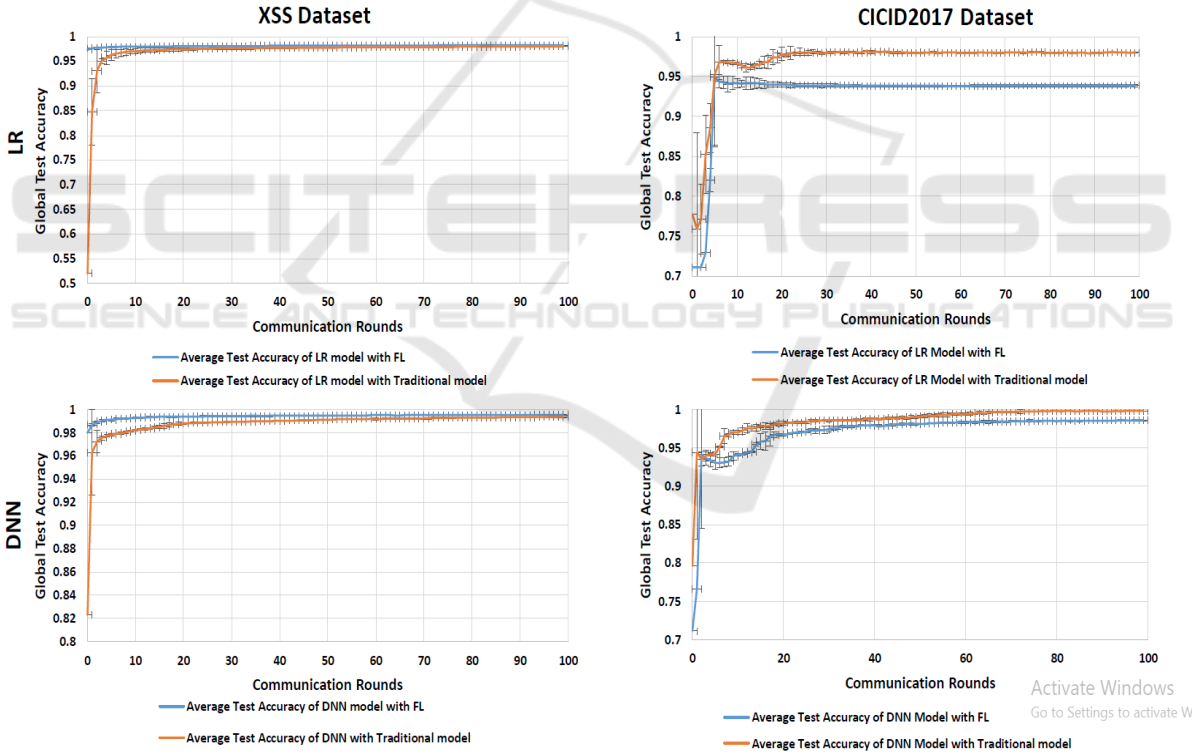

Figure 2: Federated learning with IID data distributions. The columns correspond to the XSS and CICIDS2017 datasets. The rows correspond to the models LR and DNN.

els during the communication rounds, supporting the results shown in Table 4.

Our federated learning framework also supports configurations with more than two clients. We tested the performance of the new classifiers, SVM and KNN, with a setup involving ten clients. Specifically, we assigned five clients to class 0 and the remaining five clients to class 1. In this setup, client i for $i \in \{1,\ldots,5\}$ was assigned all data points of class 0, while client j for $j \in \{6,\ldots,10\}$ was assigned all data points of class 1. This setup, detailed in Table 4, demonstrates the scalability and robustness of our approach.

FL's effectiveness tends to align with the balance

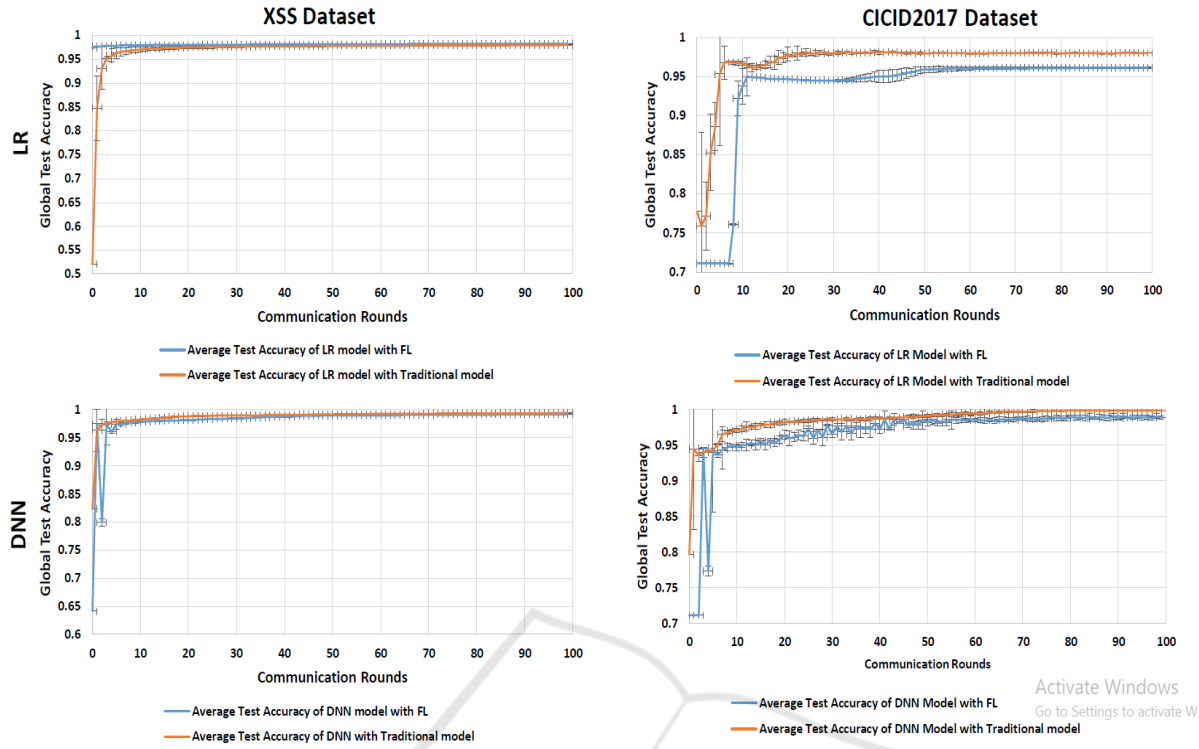

Figure 3: Federated learning with non-IID data distributions. The columns correspond to the XSS and CICIDS2017 datasets. The rows correspond to the models LR and DNN.

between model and dataset complexity. Matching the model complexity to that of the dataset is crucial to fully benefiting from the FL effect. We can evaluate our model against our proposed scheme's benchmarks provided by (Yan et al., 2022; Mereani and Howe, 2018). It is worth noting that all machine learning models referenced in the prior studies are employed as traditional centralized models. A key advantage of our approach is that with federated learning, accessing the clients' data is unnecessary.

6 CONCLUSION

Users' data privacy concerns have become more important, and traditional methods face problems safeguarding sensitive data. Federated learning with ML models can be used as a verification stage to ensure privacy while training ML models. In this research, we presented an innovative and scientifically sound privacy-preserving FL as an alternative method to a centralized model in detecting XSS attacks. Our approach facilitates the training of ML models on distributed devices, effectively mitigating the privacy risks of sensitive data. We comprehensively assessed the proposed framework using authentic, real-world data and compared its efficacy with traditional centralized ML methodologies. The experimental findings strongly indicated that the proposed FL approach attained performance levels comparable to those of centralized models such as LR and DNN while ensuring data privacy. The outcomes affirm that FL holds excellent promise as a viable technique for XSS detection. At the same time, our framework exhibits potential for adaptation to address other security vulnerabilities prevalent in web applications. Future research should aim to gain a more thorough understanding of XSS behavior. We will expand our research to include more attack types, such as SQL injection and cross-site request forgery. Moreover, we will utilize different models and investigate the complexity of these strategies.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge funding from the Koret Foundation grant for Smart Cities and Digital Living 2030.

REFERENCES

Aebersold, S., Kryszczuk, K., Paganoni, S., Tellenbach, B., and Trowbridge, T. (2016). Detecting obfuscated javascripts using machine learning. In *ICIMP 2016 the Eleventh International Conference on Internet Monitoring and Protection, Valencia, Spain, 22-26 May 2016*, volume 1, pages 11–17. Curran Associates.

Ariu, D. and Giacinto, G. (2011). A modular architecture for the analysis of http payloads based on multiple classifiers. In *Multiple Classifier Systems: 10th International Workshop, MCS 2011, Naples, Italy, June 15-17, 2011. Proceedings 10*, pages 330–339. Springer.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191.

Galán, E., Alcaide, A., Orfila, A., and Blasco, J. (2010). A multi-agent scanner to detect stored-xss vulnerabilities. In *2010 International Conference for Internet Technology and Secured Transactions*, pages 1–6. IEEE.

Hsieh, K., Phanishayee, A., Mutlu, O., and Gibbons, P. (2020). The non-iid data quagmire of decentralized machine learning. In *International Conference on Machine Learning*, pages 4387–4398. PMLR.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. (2021). Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210.

Klein, A. (2005). Dom based cross site scripting or xss of the third kind. *Web Application Security Consortium, Articles*, 4:365–372.

Komiya, R., Paik, I., and Hisada, M. (2011). Classification of malicious web code by machine learning. In *2011 3rd International Conference on Awareness Science and Technology (iCAST)*, pages 406–411. IEEE.

Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

Kotzur, M. (2022). Privacy protection in the world wide web—legal perspectives on accomplishing a mission impossible. In *Personality and Data Protection Rights on the Internet: Brazilian and German Approaches*, pages 17–34. Springer.

Lee, S., Wi, S., and Son, S. (2022). Link: Black-box detection of cross-site scripting vulnerabilities using reinforcement learning. In *Proceedings of the ACM Web Conference 2022*, pages 743–754.

Li, L., Fan, Y., Tse, M., and Lin, K.-Y. (2020a). A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. (2020b). Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450.

Likarish, P., Jung, E., and Jo, I. (2009). Obfuscated malicious javascript detection using classification techniques. In *2009 4th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 47–54. IEEE.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. (2017). Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*.

Luo, C., Su, S., Sun, Y., Tan, Q., Han, M., and Tian, Z. (2020). A convolution-based system for malicious urls detection. *Computers, Materials & Continua*, 62(1).

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Mereani, F. A. and Howe, J. M. (2018). Detecting cross-site scripting attacks using machine learning. In *International conference on advanced machine learning technologies and applications*, pages 200–210. Springer.

Meyer, R. and Cid, C. (2008). Detecting attacks on web applications from log files. *Sans Institute*.

Ndegwa, A. (2016). What is a web application. *Maxcdn [En línea]*, 31.

Nunan, A. E., Souto, E., Dos Santos, E. M., and Feitosa, E. (2012). Automatic classification of cross-site scripting in web pages using document-based and url-based features. In *2012 IEEE symposium on computers and communications (ISCC)*, pages 000702–000707. IEEE.

OWASP (2017). Owasp top 10 - 2023 rc1. `https://owasp.org`, note = Accessed on [26-9-2023],.

Rocha, T. S. and Souto, E. (2014). Etssdetector: A tool to automatically detect cross-site scripting vulnerabilities. In *2014 IEEE 13th International Symposium on Network Computing and Applications*, pages 306–309. IEEE.

Shamir, O., Srebro, N., and Zhang, T. (2014). Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008. PMLR.

Sharafaldin, I., Lashkari, A. H., Ghorbani, A. A., et al. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116.

Wei-Hong, W., Yin-Jun, L., Hui-Bing, C., and Zhao-Lin, F. (2013). A static malicious javascript detection using svm. In *Conference of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)*, pages 214–217. Atlantis Press.

Wu, D., He, Y., Luo, X., and Zhou, M. (2021a). A latent factor analysis-based approach to online sparse streaming feature selection. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(11):6744–6758.

Wu, D., Luo, X., Shang, M., He, Y., Wang, G., and Zhou, M. (2019). A deep latent factor model for high-dimensional and sparse matrices in recommender systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(7):4285–4296.

Wu, D., Shang, M., Luo, X., and Wang, Z. (2021b). An l 1-and-l 2-norm-oriented latent factor model for recommender systems. *IEEE Transactions on Neural Networks and Learning Systems*, 33(10):5775–5788.

Wu, X., Zheng, W., Chen, X., Zhao, Y., Yu, T., and Mu, D. (2021c). Improving high-impact bug report prediction with combination of interactive machine learning and active learning. *Information and Software Technology*, 133:106530.

Yan, H., Feng, L., Yu, Y., Liao, W., Feng, L., Zhang, J., Liu, D., Zou, Y., Liu, C., Qu, L., et al. (2022). Cross-site scripting attack detection based on a modified convolution neural network. *Frontiers in Computational Neuroscience*, 16:981739.

Yang, Q., Liu, Y., Chen, T., and Tong, Y. (2019). Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19.

Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., and Gao, Y. (2021). A survey on federated learning. *Knowledge-Based Systems*, 216:106775.

ZHOU, K., WAN, L., and DING, H.-w. (2019). A cross-site script detection method based on mlp-hmm. *Computer Engineering & Science*, 41(08):1413.