applied sciences

MDPI

Article

BChainGuard: A New Framework for Cyberthreats Detection in Blockchain Using Machine Learning

Suliman Aladhadh *, Huda Alwabli, Tarek Moulahi * and Muneerah Al Asqah

Department of Information Technology, College of Computer, Qassim University, Buraidah 52571, Saudi Arabia
* Correspondence: s.aladhadh@qu.edu.sa (S.A.); t.moulahi@qu.edu.sa (T.M.)

Abstract: Recently, blockchain technology has appeared as a powerful decentralized tool for data integrity protection. The use of smart contracts in blockchain helped to provide a secure environment for developing peer-to-peer applications. Blockchain has been used by the research community as a tool for protection against attacks. The blockchain itself can be the objective of many cyberthreats. In the literature, there are few research works aimed to protect the blockchain against cyberthreats adopting, in most cases, statistical schemes based on smart contracts and causing deployment and runtime overheads. Although, the power of machine learning tools there is insufficient use of these techniques to protect blockchain against attacks. For that reason, we aim, in this paper, to propose a new framework called BChainGuard for cyberthreat detection in blockchain. Our framework's main goal is to distinguish between normal and abnormal behavior of the traffic linked to the blockchain network. In BChainGuard, the execution of the classification technique will be local. Next, we embed only the decision function as a smart contract. The experimental result shows encouraging results with an accuracy of detection of around 95% using SVM and 98.02% using MLP with a low runtime and overhead in terms of consumed gas.

Keywords: blockchain; Ethereum; cyberthreats; machine learning; MLP; SVM; smart contract

Citation: Aladhadh, S.; Alwabli, H.; Moulahi, T.; Al Asqah, M. BChainGuard: A New Framework for Cyberthreats Detection in Blockchain Using Machine Learning. *Appl. Sci.* **2022**, *12*, 12026. https://doi.org/10.3390/app122312026

Academic Editor: Xiaoyang Liu

Received: 13 October 2022
Accepted: 18 November 2022
Published: 24 November 2022

1. Introduction

The concept of blockchain was firstly proposed by Satoshi Nakamoto in [1] as a decentralized system for money transfer.

Blockchain is a system that consists of a network of multiple blocks where each block contains several transactions, and each block connects to its previous block through a hash-based procedure. This hashing procedure and the utilization of other cryptography methods have given blockchain technology its property of protecting block content integrity [1]. Figure 1 shows the list of blockchain characteristics [2], which are:

- Block: contains a list of information such as the block number, nonce, time stamp, data, the hash of the previous block, and the hash of block itself.
- Ledger: a list of block forms a ledger.
- Distribution: is an important characteristic of blockchain, as blockchain architecture is based on P2P network, and each miner contains the whole blockchain.
- Transaction: the data in the block are a list of transaction
- Confirmation: is needed by at least 51% of miners to validate the list of transactions in the block.
- Proof of work: is the right value of nonce giving a block hash starting by a list of zeros.
- Result: add a new block to the blockchain.

1.1. Motivation and Problem Statement

Blockchain has attracted the research community, as well as the industry. Actually, blockchain is used in serval domains such as the Internet of Things [3,4] and also in healthcare [5,6], in addition to other fields such as finance [7].

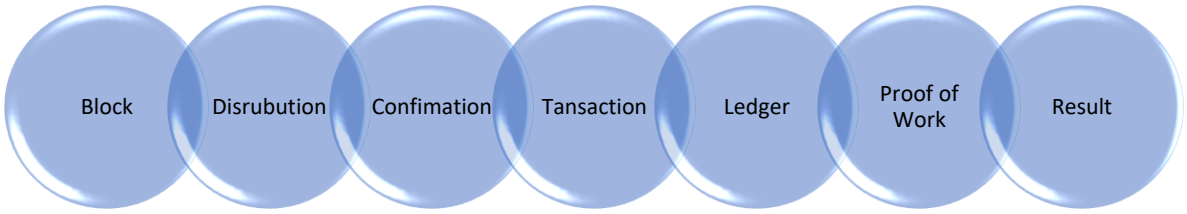

Figure 1. Blockchain characteristics.

Blockchain characteristics ensure content and operations safety, making it a suitable choice for system security, although its characteristics are the subject of many cyberthreats such as DDoS, eclipse, 51% attack, and so on. Therefore, developing a cyberthreats detection system on the blockchain is highly needed.

Although many efforts have been made by the research community to tackle blockchain threats, this domain needs more investigation. Most of the proposed framework in this context is using statistical methods or simple algorithms, but machine learning techniques are lowly used. Furthermore, there are many limitations in existing related works such as the overheads due to deployment and runtime. More details will be given later in this paper.

1.2. Objectives

The main objective of this paper is to develop a framework called BChainGuard for cyberthreats detection in Blockchain. BChainGuard will be based on smart contracts in order to assure safe execution and on machine learning tools in order to provide high-accuracy detection. This will help to distinguish between normal and abnormal behavior of the traffic linked to the blockchain network. This choice will decrease the deployment and the runtime overhead. In the proposed framework, the execution of the classification technique will be in a local machine. Next, we embed only the decision function as a smart contract. This will create a new layer in blockchain for cyberthreat detection.

The contributions of this research work can be summarized as follows:

- Build a novel framework to detect cyberthreats on blockchain by merging machine learning and deep learning to increase the accuracy of detection and the blockchain itself.
- Propose a new layer in the blockchain-based on smart contracts for cyberthreats detection in order to protect the blockchain.
- Measure the effectiveness of the proposed technique by comparing it with an existing solution [8] in terms of accuracy, deployment, and execution overhead.

1.3. Paper Organization

The rest of this paper is organized as follows: Section 2 outlines the literature review including a study of recently proposed techniques that deal with the issue of blockchain attacks and protection. Section 3 describes the BChainGuard contribution. Section 4 presents the experimental study. Finally, the conclusion is given in Section 5.

2. Literature Review

This section is divided into three subsections. First, a study on the most relevant attacks on the blockchain is given. Next, it goes through a literature review to outline the framework and system for intrusion detection in the blockchain. Finally, we discuss and compare these frameworks.

2.1. Review of Blockchain Cyberthreats

Many cyberthreats and their countermeasures have been studied by research communities. In what follows, we outline the most relevant works in this context.

In [9], the authors studied (Distributed Denial of Service) DDoS attacks. This attack can cause an increase in mining fees, and it can have an effect on the cryptocurrency

systems' memory pools. The authors use Bitcoin mempools to study DDoS and to observe its effect. To tackle DDoS attacks, the authors propose fee-based and age-based designs. The proposed technique is validated through simulation studies with different kinds of attack conditions.

In [10], the authors discussed a DAO attack. They proposed a framework called VeriSolid to be used to formally verify smart contracts, which are using a transition system-based model. This will help developers to check, at a high level of abstraction, the contract behavior. The proposed framework helps in generating the Solidity code from the verified models. This allows the development of smart contracts based on correct-by-design concept.

Another type of BC attack was studied in [11]. The authors showed that the most widely used Ethereum implementation called Go Ethereum (Geth) is exposed to eclipse attacks. For that reason, they study the fundamental properties of Geth's node discovery, which can be the cause of false friends' attacks, and they propose countermeasures to avoid the eclipse attack.

In [12], the authors proposed a countermeasure for a malleability attack. This attack happens when the malicious tries to change some byte in the signature, while it may still be effective, so that the attacker can control bitcoin transactions. To tackle this, attack the authors suggest modifying the specification of bitcoin by calculating the transaction hash deprived of its input script.

The paper of [13] dealt with time hijacking attacks that happened due to the vulnerability of the time stamp process in bitcoin. The attacker alters the bitcoin time counter, as well as the node's time. This can lead to a perturbation of times. To avoid this attack, the authors propose to not accept time ranges or not utilize the system time of nodes.

Sybil attack was discussed in [14]. This kind of attack occurs when the malicious try to control the blockchain network by creating a great number of pseudonymous identities and manipulating blockchain redundancy and anonymity. The authors proposed to avoid this attack by proving identities using a trusted agency. This credible agency is responsible for certifying identities. Before being part of a Peer-to-Peer network, a third party will be used to authenticate it.

In [15], the authors outlined the phishing attack which happens when the attacker tries to steal user credentials through websites or using untrue emails or both of them simultaneously. To defend against this attack, the authors suggest adopting a strategy based on excluding the unreasonable behavior of the user in addition to detecting and filtering phishing resources. This will help to avoid the stolen of phishing infrastructure elements such as passwords.

Border Gateway Protocol (BGP) attacks can target blockchains [16]. This attack is linked to routing protocols. It happens when a malicious system creates and broadcast fake advertisements to its neighbors. This can result in the redirection of traffic to specified destinations. In this case, the attacker may control the node traffic destroying the consensus mechanism. To defend against BGP attacks, the authors propose a new Bitcoin relay network with great extensibility and safety denoted as SABRE, where blocks are relayed using a list of connections that can resist attacks on routing.

In [17], the authors discussed selfish mining attacks. This attack occurs when selfish miners use selfish techniques for getting non-deserved incomes. As an example, the malicious miner pool decides to not disseminate the block after finding the next block and to continue the hashing process in order to create a new valid chain and neglect the right one. To tackle this attack, the authors proposed to neglect blocks that are not achieved in time and give awards to blocks merging links into their previous competing blocks.

Another serious attack targeting blockchain is called the integer overflow attack [18]. This attack is a serious problem linked to Ethereum smart contracts, which are program codes. The integer in these codes has an upper and lower limit. The integer overflow attack can happen essentially in the value-type conversion, which can cause a big loss. To tackle this kind of attack, the authors propose the Osiris framework in order to control integer overflow in a smart contract by combining taint analysis and symbolic execution.

2.2. Intrusion Detection System in Blockchain

Although blockchain has been widely used as a tool for IDS in other environments, there is an extreme lack of IDS solutions on the blockchain.

In [19], the author proposed to use a GPU solution based on TRS (Target Rooted Subgraph) to detect anomalies in transactions by using a part of the data. Through an experimental study, the proposed techniques archive 195 times faster than the existing method. Furthermore, it outperforms the existing method in terms of accuracy and true positive rate, due to the use of subgraphs to detect local anomalies in the case of transactions on a small scale. This rate is near the existing method in the case of transactions on a large scale.

The authors In [20] proposed a method to detect anomalies in bitcoin transactions. They use a dataset including a list of bitcoin transactions with normal behavior. The authors created another list of transactions with abnormal behavior by including three types of attacks, which are: DDoS, 51% attack, and Double spending attack. For the detection, they used SVM (support vector machine) and K-means. Through experimental study, they show that both techniques give good accuracy.

Since smart contracts are vulnerable to attacks, in [21], the authors discussed tackling these vulnerabilities by proposing ContractGuard as the first intrusion detection system for smart contracts in Ethereum. After deploying the proposed system with real smart contracts in Ethereum, the authors showed the effectiveness of their solution in terms of protection with low execution overheads (only add 28.27%) and with light deployment (only add 36.14%). The vulnerabilities decreased by a rate of 83%.

In [22], the authors proposed SODA as a new framework for the online detection of many attacks. The authors showed that SODA outperforms existing solutions in terms of efficiency, compatibility, and capability. The aim of SODA is to let users rapidly develop a new application for cyberthreat detection. To show the effectiveness of SODA, the authors developed 8 applications, including new detection methods for the detection of attacks focusing on smart contract vulnerabilities. SODA is also embedded into the EVM-based blockchain. Through experiments, SODA shows effective attack detection with low overhead.

Another proposed IDS was denoted BAD: Blockchain Anomaly Detection [23]. BAD can detect only two types of attack: eclipse attack and zero-day attack. The main goal of BAD is to detect anomalies in transactions in addition to preventing their spreading. The prevention is based on the collection of malicious activities in addition to building a distributed database of threats. The authors make an analysis of BAD overhead, its implementation, and in order to show its effectiveness in the detection of eclipse attacks, as well as zero-day attacks.

Recently, an important technique called SolGuard was proposed in [24] in order to prevent all issues linked to smart contracts. The proposed work is based on using multi-agent robotic systems. The work is based on studying Ethereum smart contracts to show its vulnerabilities due to several programming problems—in particular, the use of low-level calls to malicious resources. The proposed technique is performed by implementing SolGuard, aiming to prevent three serious issues linked to low-level calls to malicious resources done by smart contracts written in solidity. Through an empirical study, based on efficiency and accuracy, the proposed technique is outperforming existing tools in the same context.

In [25], DefectChecker was proposed aiming to detect defects in smart contracts by using symbolic execution to analyze the bytecode of the smart contract. In fact, DefectChecker is using various rules in order to detect eight vulnerabilities in contracts. After running it on a previous work dataset, it accomplishes encouraging results. DefectChecker outperforms some existing tools for defect detection in smart contracts. The experiment shows that 15.89% of Ethereum smart contracts include at least one example of the eight vulnerabilities in the contract.

2.3. Discussion

In this subsection, we make a discussion of the previously outlined system and framework for intrusion and threats detection in the blockchain. Table 1 presents a general comparison of them.

Table 1. Comparison of the existing frameworks for blockchain intrusion detection.

Tools	Technique of Detection	Cyberthreat Coverage	Limitations
TRS [17]	Using graph theory to detect anomalies in smart contract	Smart contract vulnerabilities	In the case of a transaction having a large scale this rate of anomalies detection is similar to the existing method
ADM [18]	Using one-class SVM and K-means for transaction anomalies detection	DDoS, 51% attack, and Double spending attack	The anomalies are executed by the authors themselves so they can be nonrealistic
ContractGuard [19]	A mechanism for intrusion detection	Attempts of malicious intrusion	Needs more experience with real smart contract vulnerabilities
SODA [20]	On-chain applications to detect anomalies	DAO attack, time hijacking, smart contract vulnerabilities	Used only for online detection of a specific attack list
BAD [21]	Blockchain Anomaly Detection solution	Eclipse attack, zero-day attack	Used only for online detection of 2 kinds of attacks
SolGuard [22]	A solution to detect anomalies in smart contract	Smart contract vulnerabilities and DoS attack	High overheads due to deployment and runtime
DefectChecker [23]	Analyzing byte code to detect anomalies in smart contracts	Detecting if a contract is controlled by an attacker	functions call issues cannot be detected

Most of the proposed techniques to deal with attacks on BC are using statistical methods or simple algorithms, but machine learning techniques are lowly used. There are many limitations in existing related work such as the overheads due to deployment and runtime. There is a need for more simulations to measure the effectiveness of some proposed techniques. Some attacks are difficult to be detected, such as 51% attack, because it is linked to the consensus protocols themselves. Some other solutions are designed only for the Ethereum platform by using smart contracts. Consequently, the development of new techniques to detect abnormal behavior in the blockchain is still needing investigation. For that reason, it is important to develop a framework for cyberthreat detection in Blockchain. This framework will be based on smart contracts (to guarantee safe execution) and on machine learning tools (to ensure high-accuracy detection). BChainGuard's main goal is to distinguish between normal and abnormal behavior of the traffic linked to the BC network. To significantly decrease the deployment and the runtime overhead, our framework will be a hybrid. The execution of the classification technique will be done locally. Next, we embed only the decision functions as smart contracts. This will create a new layer in blockchain for cyberthreat detection.

3. Contribution: BChainGuard

In this section, we describe the proposed framework in Figure 2 in addition to the list of used algorithms to detect cyberthreats.

3.1. BChainGuard Phases

Phase 1: Selection of the dataset and artificial intelligence tool

In this phase, the dataset of attack on the blockchain is selected which is located in [26]. In addition, we plan to use SVM (support vector machine) as a machine learning tool and MLP (multi-layer perceptron) as a deep learning tool.

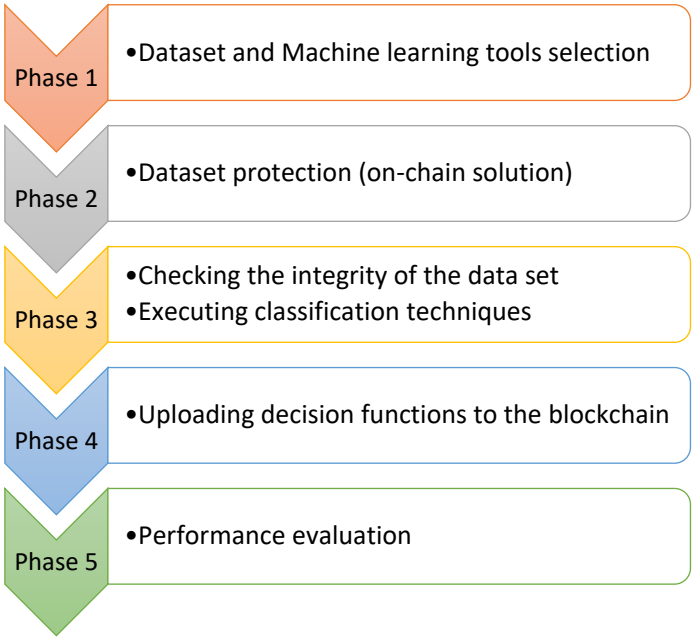

Figure 2. Overview of the proposed work.

Phase 2: Linking the dataset to the Ethereum blockchain

To link the dataset from the client to the blockchain, there are two strategies: (1) on-chain solution, which means uploading the whole dataset in the blockchain, or (2) off-chain solution, which means uploading only the hash of the dataset to the blockchain. Both strategies help to protect the integrity of the dataset.

Phase 3: Checking the integrity of the data set and executing classification techniques

Before executing the classification techniques on the client side using Python, we can either: (1) download the dataset from the blockchain in case we choose on-chain strategies or (2) hash the dataset and compare the hash with that of the dataset in the blockchain in case of off-chain strategies.

Phase 4: Uploading decision functions to the blockchain

During this phase, classification techniques are executed in the local machine using Python. Next, we extract the parameters of the decision function for both SVM and MLP. Finally, upload these parameters to be used by smart contracts on blockchain to segregate between normal behavior and abnormal behavior. This helps to protect the blockchain against attack.

Phase 5: Performance evaluation

The performance evaluation of the proposed techniques can be measured following many parameters, such as the accuracy, the recall, the f1-score, and the time of execution. The proposed technique will also be compared with an existing technique using the same dataset [27] to validate the contribution and to show its effectiveness.

3.2. SVM Training and Parameters Extraction

To train SVM, we used the Radial Basis Kernel Function (RBF). This function calculates the Euclidean distance between vectors. After the training process was completed, the training parameters of the following model were extracted to be kept on-chain, as shown in Table 2.

Table 2. Extracted SVM training parameters.

Name	Description	Size	Data Type
support_vectors_	Datapoints defining hyperplane decision boundaries placements	18×395	Decimal matrix
_dual_coef_	Weights of data points	1×395	Decimal array
intercept	The bias	1	Decimal
_gamma	The parameter that handles non-linear classification	1	Decimal

Algorithm 1 describes the steps of SVM execution and the decision parameter extractions. Those parameters will be sent next to the blockchain to be used for the decision function.

Algorithm 1: SVM execution

Input: *dataset*
Output: *decision parameters*
1: Read dataset
2: Read dataset hash from the blockchain
3: Check dataset authenticity
4: Execute SVM on the dataset
5: Print SVM performances Accuracy, Precision, F1-score, Recall
6: Extract decision parameters
7: Send decision parameters to smart contract
8: Print Consumed Gaz, Time

3.3. MLP Training and Parameters Extraction

The MLP training process included two hidden layers of sizes 5 and 2 over 1000 epochs. The feedforward deep learning NN relied on a nonlinear activation function, known as the Rectifier Linear Unit activation function (ReLu). Indeed, Table 3 shows the MLP parameters stored on-chain after the training is complete.

Table 3. MLP parameter details.

Name	Description	Size	Data Type
coefs_	Weights of neuron's inputs in three layers, two input layers, and the output layer	5×18 2×5 1×2	Decimal matrices
intercepts_	Biases of each neuron in three layers	1×5 1×2 1×1	Decimal arrays

Algorithm 2 describes the steps of MLP execution and the decision parameter extractions. Those parameters will be sent next to the blockchain to be used for the decision function.

Algorithm 2: MLP execution

Input: *dataset*
Output: *decision parameters*
1: Read dataset
2: Read dataset hash from the blockchain
3: Check dataset authenticity
4: Execute MLP on dataset
5: Print MLP performances Accuracy, Precision, F1-score, Recall
6: Extract decision parameters
7: Send decision parameters to smart contract
8: Print Consumed Gaz, Time

3.4. SVM Decision Function

The values returned by the decision function of an RBF kernel SVM vary between -1 and 1. This function is described in the mathematical representation below.

$$h^*(\mathrm{x}) = (\mathrm{w}^{\bigstar}\phi(\mathrm{x})) + w_0^{\bigstar} = \sum_{i \in \mathcal{P}_S} \alpha_i^{\bigstar} u_i \cdot \mathcal{K}(\mathrm{x}_i - \mathrm{x}) + w_0^{\bigstar}$$

where:

- h^* is the decision function;
- ${\alpha_i}^{\bigstar}$ is the value of the coefficients;
- u_i is the support vector output of the kernel function $\mathcal{K}$;
- x is the new data point;
- x_i is the support vector;
- $w_0^{\bigstar}$ is the bias or the intercept of each vector.

$$\mathcal{K}(\mathrm{x}, \mathrm{x}') = \exp(-\gamma \|\mathrm{x} - \mathrm{x}'\|^2)$$

The core function of this SVM implementation was the RBF function. Its mathematical representation is described above.

- The RBF kernel function returns the product of negative gamma with the Frobenius norm of two input vectors.
- The exp() function is the exponent of Euler's number, e.

$$\|\mathrm{x} - \mathrm{x}'\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|^2}$$

The above math representation shows that the Frobenius norm is the square root of the summation of two input vectors' squared difference. In this study's implementation case, the two input vectors are the new data point and the support vector.

Algorithm 3 shows the steps taken to compute the decision function of an RBF kernel SVM using the previous procedures.

Algorithm 3: SVM decision function for attack detection

Input: *V: input vector including list of features*
Output: *attack or normal behavior*
Read SVM decision parameters from Python
Create decision function based on RBF kernel and using decision parameters
Execute decision function based on the input vector and get the result
If the result is near -1 **Then**
    Normal behavior
Else
    Attack
End If

3.5. MLP Decision Function

MLP is a supervised classifier that takes an n-dimensional input to return an m-dimensional output.

In the implementation, the input layer of MLP is 9-dimensional, and the two hidden layers 5 and 2-dimensional. Additionally, there is a 1-dimensional output layer. Figure 3 shows a visualization of the implemented MLP layers.

Where:

- x denotes the input characteristics;
- a designates the neurons of the first hidden layer;

- p designates the neurons of the second hidden layer;
- b denotes the bias;
- y denotes the output neuron.

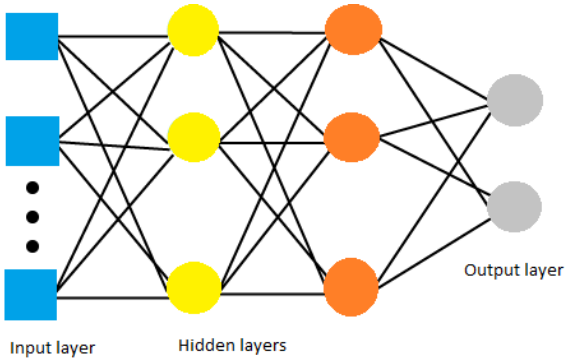

Figure 3. MLP layers visualization.

MLP's decision function concludes a series of additions and multiplications to classify an input. In this calculation, the value of each hidden neuron is equal to the linear sum of all the neuron values of the previous layer multiplied by their coefficients, knowing that the weights between the neuron's layer and the last layer.

An additional intercept value, or bias, is added to this summation.

where:

- x is the value of the neuron;
- w is the weight of the neuron.

Note that the bias, b, is denoted by x_0 with a value of +1, and the w_0 is the intercept value in each layer's bias table.

$$h_i^{(1)} = \phi\left(\sum_j x_j w_{ij}^{(1)} + b_i^{(1)}\right)$$

$$h_i^{(2)} = \phi\left(\sum_j h_j^{(1)} w_{ij}^{(2)} + b_i^{(2)}\right)$$

$$y_i = \phi\left(\sum_j h_j^{(2)} w_{i,j}^{(3)} + b_i^{(3)}\right)$$

In the above representation, h in is the neuron, i, value in the nth layer. This implementation includes two layers and a final output layer with one neuron,

- y_i, gives the final summation value.
- $\phi()$ is the non-linear activation function that calculates neuron's value by a weighted sum.
- x_j is the input features vector.
- $h_j{}^n$ is the neurons' values at layer $i-1$.
- $w_{i,j}$ is the weight.
- $b_i{}^n$ is the intercept of neuron i at the nth layer.

$$f(x) = \max(0, x)$$

This study employed the ReLu nonlinear activation function. The above annotation shows that the ReLu function returns the maximum between x and 0. Algorithm 4 shows steps of implementing the MLP decision function.

Algorithm 4: MLP decision function for attack detection

Input: *V: input vector including list of features*
Output: *attack or normal behavior*
Read MLP decision parameters from Python
Create decision function using decision parameters
Execute decision function based on the input vector and get the result
If the result is −1 **Then**
    Normal behavior
Else
    Attack
End If

4. Experimental Results

In this section, we give the description for the dataset and the environment of execution. We also discuss the experimental results.

4.1. Dataset Description

This study implementation used the dataset stored in Github [27]. It is a website that makes it easy for programmers and developers to work together to improve application code. It relies on the principle of version control, which uses branching and merging to ensure seamless collaboration without affecting the integrity of the original project [26]. In Table 4, we present the used dataset that contain a list of attack on the Ethereum blockchain.

Table 4. Dataset features.

	Features	Features Details
Old features	hash	Transaction hash
	nonce	How many transactions did the sender's account make?
	transaction indicator	Transaction index in block
	From the address	Origin account
	To the address	destination account
	The value	The transferred value in Wei which is the smallest Ether unit
	gas	Quantity of gas per source
	gas_price	The price of gas (Wei) which is provided by the source
	input	The data sent during the transaction
	cumulative gas reception used	How much gas was used by this transaction while executing a block
	receiving gas used	Total gas has been used by this single given transaction
	timestamp_block	Block timestamp was used by this transaction
	block_number	Operation block number
	block hash	Hashing the block used during the transaction
New added features	Of fraud	- 1 indicates that the return address is the result of forgery - 0 indicates that the sender's address is correct
	to sheat	- 1 indicates that the return address is the result of forgery - 0 indicates that the sender's address is correct
	from_category	Determine if the abnormal activity that occurred from the sender address is phishing or scamming, and (null) for normal operation
	to_category	Determine if the abnormal activity that occurred from the sender address is phishing or scamming, and (null) for normal operation

To evaluate the detection systems that use labeled data (i.e., transactions), the proposed data set should be labeled. Therefore, the transactions included in the proposed data set were classified as normal or harmful transactions, so the number "0" indicates that the

transaction is valid., while the number "1" means that the transaction is an attack. As a reminder, each transaction has two addresses: (1) for the sender and (2) for the recipient.

The address account has been passed to the Etherscamdb API through the Python programming language to find out the source of the fraud. Is it the sender or the receiver? Then, four new columns were added to the transaction table: "of_fraud", "to sheat ", "from_category", and "to_category"; the description of these extensions is found in Table 4.

Additionally, two types of attacks were recorded in relation to the proposed technique when API response, as there are two main categories, namely phishing and scamming, as shown in Table 5 and Figure 4. The percentages of abnormal transactions are 22% and 80%, respectively.

Table 5. Ethereum transactions in the dataset.

Type	Transaction	Ratio
Threat transaction	14,250	20%
Normal transaction	57,000	80%
Total	71,250	100%

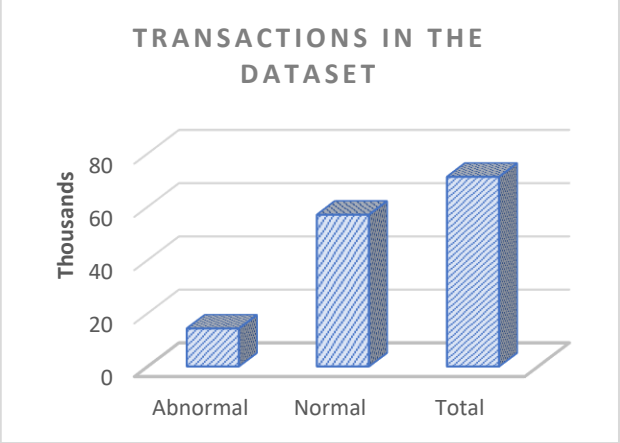

Figure 4. Distribution of transactions in the dataset.

4.2. Environment Description

Table 6 shows the development computer system and the specifications of the development tools.

Table 6. System specifications.

Item	Specifications
Computer OS	Windows 10
CPU	AMD64 3.20 GHz
RAM	8 GB
Ganache	v2.5.4
Solidity	v0.5.16
Vyper	v0.3.0
MetaMask	v10.2.2
Python	v3.9.7

The experimentation is done as follows:

- We use a smart contract to upload the dataset line by line to Ethereum (Ganache) for integrity protection.

- SVM and MLP are executed locally on Python on the same dataset. Next, the decision parameters are extracted and sent to Ethereum (Ganache).
- We use smart contracts to receive the decision parameters on Ethereum (Ganache).
- We re-write the decision MLP decision function and SVM decision function as a smart contract on Ethereum (Ganache).
- Finally, we test the performance of MLP and SVM decision functions.

The aim of BChainGurad is to embed the protection on Ethereum (Ganache). BChainGuard is using smart contracts which work as a defender by checking the traffic of transactions to decide if abnormal behavior is detected or not based on the training already done.

In the experiment, we use Ganache as a local Ethereum platform. Solidity and Vyper are used to write the smart contract for decision function and to protect the dataset. MetaMask is a gateway allowing the communication between web applications and blockchain using 'web3.js'. The local training and testing of SVM and MLP are executed using Python. The right values of each item used in the experiment are indicated in Table 6.

4.3. Evaluation Parameters

The evaluation parameters are described by the following equation: The accuracy is in Equation (1)

$$\text{Accuracy} = (TP + TN)/(TP + FP + FN + TN) \quad (1)$$

The equation of precision is given in the following:

$$\text{Precision} = TP/(TP + FP) \quad (2)$$

Equation (3) describes the recall:

$$\text{Recall} = TP/(TP + FN) \quad (3)$$

Finally, the F1-score is described using the equation below:

$$\text{F1 Score} = 2 \times (\text{Recall} \times \text{Precision})/(\text{Recall} + \text{Precision}) \quad (4)$$

4.4. Machine Learning Result Analyses

We compared both models on the same dataset. Table 7 and Figure 5 show the performance of the MLP and SVM models. We concluded from this comparison that the two models have distinct performances. In addition, our analysis also showed that the MLP belonging to the deep learning family outperforms SVM in all metrics with a small difference (3.02% for accuracy, 3.27% for precision, 5.03% for F1-score and 3% for the recall metric). In summary, after conducting several experiments, the MLP and SVM networks have confirmed that they have very distinct solutions for regression, classification and prediction tasks.

Table 7. SVM vs. MLP performance.

Classifier	Accuracy	Precision	F1-Score	Recall
SVM	95	95.03	93	95
MLP	98.02	98.5	98.03	97

Although MLP has the best generalizability, the observed performance differences are negligible in most cases. The main difference lies in the complexity of the networks. An MLP network that implements a global approximation strategy typically uses a very small number of hidden neurons.

4.5. Blockchain Decision Function Result Analyses

Gas fees are payments made by users to pay validators and miners for the computational energy consumed to process and validate transactions on the Ethereum blockchain.

Figure 6 shows a graphical representation of the performance of MLP and SVM models in term of consumed gas in gwei. Figure 7 shows a graphical representation of the performance of MLP and SVM models in term of decision function elapsed time in seconds.

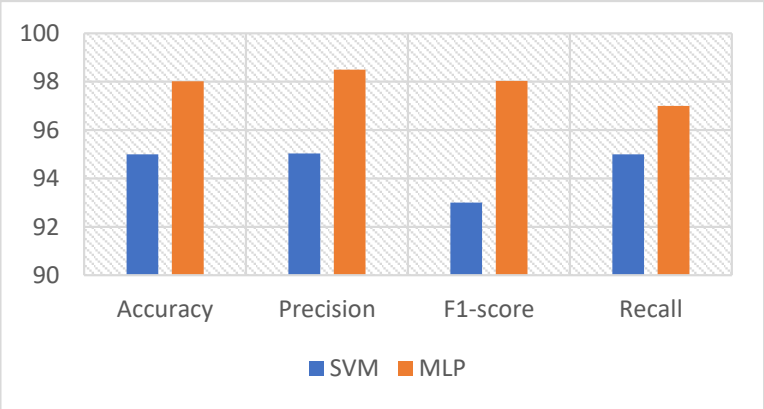

Figure 5. SVM vs. MLP performance.

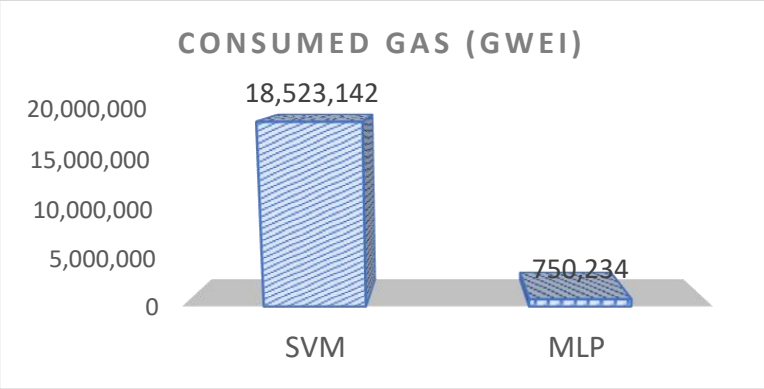

Figure 6. Consumed gas in gwei for SVM decision function and MLP decision function.

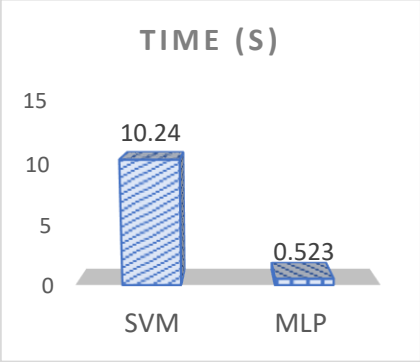

Figure 7. Elapsed time in seconds for SVM decision function and MLP decision function.

This comparison analysis is useful to verify the effectiveness of both models. The consumed gas in gwei for SVM decision function is equal to 0.1914, while this consumption is multiplied more than 44 times for SVM and reaches 8.4415. This big difference can be explained in two ways.

First explanation: the implementation of the MLP approximative strategy usually employs a very small number of hidden neurons. On the other side, the SVM is based on the local approximation strategy and uses a large number of hidden units. This large number can be the cause of differences in consumption and also need more time for execution.

Second explanation: If a gas price is set too low, the transaction could be ignored, missed or the wallet could become stuck, freezing transactions from that wallet. Therefore, the wallet can remain blocked until the transaction is resolved.

4.6. Overeall Result Discussion

An overall result comparison shows that MLP is providing the best results. Indeed, the SVM decision function is more expensive in terms of consumed gas and time. This can be explained by the number of parameters and operations in SVM decision function. In our case, we adopt only two layers in MLP. For that reason, the MLP decision function is cheaper than that of SVM.

Furthermore, MLP outperforms SVM in terms of accuracy and precision; this can be rendered to the data itself. In addition, these results are linked to the implementation of the MLP as an approximative strategy that usually employs a very small number of hidden neurons while SVM is based on the local approximation.

The provided performances' favorite MLP is used in this case. However, this is always linked to the type of the used transaction themselves. An enlargement of the dataset by considering more types of attacks may favorite another technique to be used for cyberthreat detection.

4.7. Comparison between BChainGuard and Works on the Same Dataset

In this section, we make a comparison between our framework result and the result of work on the same dataset. In BChainGuard, the accuracy of MLP in is 98.2% and 95% for SVM. However, in [28], the best accuracy is performed at 98.8% using Random Forest, while the smallest one is 82% for logistic regression. In the same context, the accuracy of detection abnormal transaction using SVM and KNN is 95%. Our contribution is not only improving the accuracy of detection but also securing the dataset against poisonous attack, making the detection safe by embedding the decision function as a smart contract. In addition, we chose to embed only the decision function on the blockchain to minimize the overhead of runtime and deployment, which was discussed in the previous section. In what follows, we make a comparison between BChainGuard and other frameworks.

Table 8 shows that our framework is the first one that uses machine learning, embedded in the blockchain itself, and with the lowest deployment and runtime overhead compared to others. In future works, we plan to make a secure analysis of BChainGuard by injecting attacks on the dataset, as well as on Ethereum, to see the performance of our framework.

Table 8. Frameworks comparison.

Tools	Technique	Place	Overhead
BChainGuard	Machine learning	On-chain	Low
TRS [17]	Graph theory	On-chain	High
ADM [18]	Machine learning	off-chain	-
ContractGuard [19]	Statistical analysis	On-chain	High
SODA [20]	Statistical analysis	On-chain	High
BAD [21]	Statistical analysis	On-chain	High
SolGuard [22]	Statistical analysis	On-chain	High
DefectChecker [23]	Statistical analysis	On-chain	High

Recently, many research efforts have been conducted to show the great impact of applying blockchain in Industry 4.0 [28]. This generation of industry investigates the application of the latest technology innovation in Artificial Intelligence, Big Data, the Internet of Things and Blockchain for supply chain and manufacturing improvement. On one hand, Blockchain with its great potential can raise many opportunities for Industry 4.0. On the

other hand, although Blockchain is one of the most secure peer-to-peer systems, it can be the target of attacks and cyberthreats. Consequently, securing service-based blockchain has high importance. In this context, we believe that BChainGuard is a successful key to supporting the application of Blockchain in Industry 4.0. Our framework can be improved by considering more types of attacks and next integrated into Ethereum as a new protection layer against smart contract vulnerabilities. This can improve Ethereum's trustworthiness to be more attractive to the industry.

4.8. BChainGuard Limitations and Possible Future Improvements

BChainGuard is based on executing SVM and MLP locally and next embedding only the decisions function in blockchain in order to detect attacks. Despite the advantages of our contribution, it can be improved in the future by:

- Considering more realistic datasets linked to blockchain smart contract and transactions.
- Using different scenarios of transactions and smart contracts that help to convert more situations that can be the target of attacks.
- Applying other machine learning and deep learning techniques that may be offered the best results.
- Adopting federated learning instead of machine learning in the case of the nonavailability of the dataset, since federated learning helps to protect privacy.

5. Conclusions

The blockchain, as any system, and despite of its power protection, can be prone to many attacks. For that reason, many efforts have been made by the research community to protect it. The majority of efforts have been based on analyzing statistically the smart contracts. To the best of our knowledge, we propose, for first time BChainGuard, a new layer in Ethereum for blockchain protection. Our idea is based on executing SVM and MLP locally and next embedding only the decisions function in the blockchain in order to detect attacks on blockchain. Our contribution is not only improving the accuracy of detection but also securing the dataset against poisonous attack. In addition, we choose to embed only the decision function on the blockchain to minimize the overhead of runtime and deployment.

Smart contracts can contain numerous security vulnerabilities, such as reentrancy, unhandled exceptions, Integer Overflow and unrestricted action. The aim of BChainGuard is to ensure the safe execution of smart contracts by avoiding vulnerabilities. As an open system, Ethereum can be the target of many attacks embedded in smart contracts themselves. New types of attacks can always appear. To tackle them, a strategy of BChainGuard continuous improvement must take place. This strategy must be defined to take into consideration the integration of new attack detection.

The limitations of BChainGuard are: (1) detecting only if an attack is happened or not without preventing the type of attack, (2) dealing only with two types of attacks in the used dataset, which are phishing and scamming attacks, and (3) using only MLP and SVM techniques, when other techniques may provide best detection accuracy.

Despite the advantages of our contribution, it can be improved in the future by considering more realistic datasets linked to blockchain smart contracts and transactions. Additionally, using different scenarios of transactions and smart contracts helps to convert more situations that can be the target of attacks. The use of other machine learning and deep learning techniques may offer the best results. As a future work, we also plan to make a secure performance analysis of BChainGuard. This can be done by injecting poisonous attacks on the dataset to see how blockchain can protect its integrity. In addition, we can run some malicious smart contracts on Ethereum to measure the reaction of our framework.

Author Contributions: Conceptualization, T.M.; methodology, H.A.; software, M.A.A.; validation, T.M.; formal analysis, S.A.; investigation, H.A.; resources, H.A.; data curation, M.A.A.; writing—original draft preparation, T. M.; writing—review and editing, S.A.; visualization, S.A.; supervision, T.M. and project administration, T.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Deputyship for Research& Innovation, Ministry of Education, Saudi Arabia, through project number QU-IF-4-4-1-31851.

Data Availability Statement: The used dataset was downloaded from (https://github.com/salam-ammari/Labeled-Transactions-based-Dataset-of-Ethereum-Network (accessed on 31 May 2022)). No ethics approval was required for this dataset.

Acknowledgments: The authors extend their appreciation to the Deputyship for Research& Innovation, Ministry of Education, Saudi Arabia for funding this research work through the project number (QU-IF-4-4-1-31851). The authors also thank to Qassim University for technical support.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 15 April 2022).
2. Alsayegh, M.; Moulahi, T.; Alabdulatif, A.; Lorenz, P. Towards Secure Searchable Electronic Health Records Using Consortium Blockchain. *Network* **2022**, *2*, 239–256. [CrossRef]
3. Samaniego, M.; Deters, R. Blockchain as a Service for IoT. In Proceedings of the IEEE International Conference on Internet of Things, Honolulu, HI, USA, 25–30 June 2017.
4. Alfrhan, A.; Moulahi, T.; Alabdulatif, A. Comparative study on hash functions for lightweight blockchain in Internet of Things (IoT). *Blockchain Res. Appl.* **2021**, *2*, 100036. [CrossRef]
5. AlAsqah, M.; Moulahi, T.; Zidi, S.; Alabdulatif, A. Leveraging Artificial Intelligence in Blockchain-Based E-Health for Safer Decision Making Framework. 2022. Available online: https://europepmc.org/article/ppr/ppr501665 (accessed on 15 April 2022).
6. Dubovitskaya, A.; Xu, Z.; Ryu, S.; Schumacher, M.; Wang, F. Secure and Trustable Electronic Medical Records Sharing using Blockchain. In Proceedings of the AMIA 2017 Annual Symposium Proceedings, Washington, DC, USA, 4–8 November 2017; pp. 650–659.
7. Eyal, I. Blockchain Technology: Transforming Libertarian Cryptocurrency Dreams to Finance and Banking Realities. *Computer* **2017**, *50*, 38–49. [CrossRef]
8. Al-E'mari, S.; Anbar, M.; Sanjalawe, Y.; Manickam, S. A Labeled Transactions-Based Dataset on the Ethereum Network. In *Advances in Cyber Security. ACeS 2020. Communications in Computer and Information Science*; Anbar, M., Abdullah, N., Manickam, S., Eds.; Springer: Singapore, 2021; Volume 1347. [CrossRef]
9. Saad, M.; Thai, M.T.; Mohaisen, A. POSTER: Deterring ddos attacks on blockchain-based cryptocurrencies through mempool optimization. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, Incheon, Republic of Korea, 4–8 June 2018; pp. 809–811.
10. Mavridou, A.; Laszka, A.; Stachtiari, E.; Dubey, A. VeriSolid: Correct-by-design smart contracts for Ethereum. In Proceedings of the International Conference on Financial Cryptography and Data Security, Frigate Bay, St. Kitts and Nevis, 18–22 February 2019; Springer: Cham, Switzerland, 2019; pp. 446–465.
11. Henningsen, S.; Teunis, D.; Florian, M.; Scheuermann, B. Eclipsing Ethereum Peers with False Friends. *arXiv* **2019**, arXiv:1908.10141. [CrossRef]
12. Andrychowicz, M.; Dziembowski, S.; Malinowski, D.; Mazurek, Ł. Fair two-party computations via bitcoin deposits. In *Financial Cryptography and Data Security*; Böhme, R., Brenner, M., Moore, T., Smith, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; pp. 105–121.
13. Apostolaki, M.; Zohar, A.; Vanbever, L. Hijacking bitcoin: Routing attacks on cryptocurrencies. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 375–392.
14. Swathi, P.; Modi, C.; Patel, D. Preventing Sybil Attack in Blockchain using Distributed Behavior Monitoring of Miners. In Proceedings of the 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 6–8 July 2019; pp. 1–6. [CrossRef]
15. Andryukhin, A.A. Phishing attacks and preventions in blockchain based projects. In Proceedings of the 2019 International Conference on Engineering Technologies and Computer Science, EnT, Moscow, Russia, 26–27 March 2019; pp. 15–19.
16. Apostolaki, M.; Zohar, A.; Vanbever, L. Hijacking bitcoin: Large-scale network attacks on cryptocurrencies. *arXiv* **2016**, arXiv:1605.07524.
17. Zhang, R.; Preneel, B. Publish or Perish: A Backward-Compatible Defense Against Selfish Mining in Bitcoin. In *Topics in Cryptology—CT-RSA 2017*; Handschuh, H., Ed.; Springer International Publishing: Cham, Switzerland, 2017; pp. 277–292. [CrossRef]

18. Torres, C.F.; Schütte, J.; State, R. Osiris: Hunting for integer bugs in Ethereum smart contracts. In Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC'18, San Juan, Puerto Rico, 3–7 December 2018; ACM: New York, NY, USA, 2018; pp. 664–676.
19. Morishima, S. Scalable anomaly detection method for blockchain transactions using GPU. In Proceedings of the 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), Gold Coast, Australia, 5–7 December 2019; pp. 160–165.
20. Sayadi, S.; Rejeb, S.B.; Choukair, Z. Anomaly detection model over blockchain electronic transactions. In Proceedings of the 15th International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 895–900.
21. Wang, X.; He, J.; Xie, Z.; Zhao, G.; Cheung, S. ContractGuard: Defend Ethereum Smart Contracts with Embedded Intrusion Detection. *IEEE Trans. Serv. Comput.* **2019**, *13*, 314–328. [CrossRef]
22. Chen, T.; Cao, R.; Li, T.; Luo, X.; Gu, G.; Zhang, Y.; Liao, Z.; Zhu, H.; Chen, G.; He, Z.; et al. SODA: A Generic Online Detection Framework for Smart Contracts. In Proceedings of the 27th Network and Distributed System Security Symposium, NDSS, San Diego, CA, USA, 23–26 February 2020. [CrossRef]
23. Signorini, M.; Pontecorvi, M.; Kanoun, W.; Di Pietro, R. BAD: A Blockchain Anomaly Detection Solution. *IEEE Access* **2020**, *8*, 173481–173490. [CrossRef]
24. Praitheeshan, P.; Pan, L.; Zheng, X.; Jolfaei, A.; Doss, R. SolGuard: Preventing external call issues in smart contract-based multi-agent robotic systems. *Inf. Sci.* **2021**, *579*, 150–166. [CrossRef]
25. Chen, J.; Xia, X.; Lo, D.; Grundy, J.; Luo, X.; Chen, T. DefectChecker: Automated Smart Contract Defect Detection by Analyzing EVM Bytecode. *IEEE Trans. Softw. Eng.* **2021**, *48*, 2189–2207. [CrossRef]
26. Chacon, S.; Straub, B. *Pro Git: Everything You Need to Know About Git*, 2nd ed.; Apress: New York, NY, USA, 2014.
27. Dataset. Available online: https://github.com/salam-ammari/Labeled-Transactions-based-Dataset-of-Ethereum-Network (accessed on 31 May 2022).
28. Javaid, M.; Haleem, A.; Singh, R.P.; Khan, S.; Suman, R. Blockchain technology applications for Industry 4.0: A literature-based review. *Blockchain Res. Appl.* **2021**, *2*, 100027. [CrossRef]