EtherShield: Time-interval Analysis for Detection of Malicious Behavior on Ethereum

BOFENG PAN and NATALIA STAKHANOVA, Department of Computer Science, University of Saskatchewan, Canada
ZHONGWEN ZHU, Ericsson, Canada

Advances in blockchain technology have attracted significant attention across the world. The practical blockchain applications emerging in various domains, ranging from finance, healthcare, and entertainment, have quickly become attractive targets for adversaries. The novelty of the technology coupled with the high degree of anonymity it provides made malicious activities even less visible in the blockchain environment. This made their robust detection challenging.

This article presents EtherShield, a novel approach for identifying malicious activity on the Ethereum blockchain. By combining temporal transaction information and contract code characteristics, EtherShield can detect various types of threats and provide insight into the behavior of contracts. The time-interval-based analysis used by EtherShield enables expedited detection, achieving comparable accuracy to other approaches with significantly less data. Our validation analysis, which involved over 15,000 Ethereum accounts, demonstrated that EtherShield can significantly expedite the detection of malicious activity while maintaining high accuracy levels (86.52% accuracy with 1 hour of transaction history data and 91.33% accuracy with 1 year of transaction history data).

CCS Concepts: • **Security and privacy** → **Distributed systems security**; • **Computing methodologies** → **Machine learning;**

Additional Key Words and Phrases: Blockchain, security

ACM Reference format:
Bofeng Pan, Natalia Stakhanova, and Zhongwen Zhu. 2024. EtherShield: Time-interval Analysis for Detection of Malicious Behavior on Ethereum. *ACM Trans. Internet Technol.* 41, 1, Article 2 (January 2024), 30 pages.
https://doi.org/10.1145/3633514

1 INTRODUCTION

In August 2021, the cryptocurrency platform Poly Network was hit by a major cyberattack that resulted in $611 million worth of digital tokens being stolen, the biggest loss in crypto ecosystem to date [1]. The attacker exploited a vulnerability in a smart contract allowing them to successfully transfer the tokens to their own accounts. Although most of the assets were eventually returned, the process of recovering the loss has emphasized a dire need for strong blockchain security guarantees.

Authors' addresses: B. Pan and N. Stakhanova, 176 Thorvaldson Bldg 110 Science Place, Department of Computer Science, University of Saskatchewan Saskatoon, SK S7N 5C9 Canada; e-mails: panbofeng@hotmail.com, natalia@cs.usask.ca; Z. Zhu, Ericsson, 8275 Trans Canada Route, Saint-Laurent, Quebec H4S 0B6 Canada; e-mail: zhongwen.zhu@ericsson.com.

1533-5399/2024/01-ART2 $15.00
https://doi.org/10.1145/3633514

Blockchain technology is getting increasingly popular across industries that require transparency and trust. There are over 1,000 blockchain platforms available in the world today [2]. Many of them enable the creation and automated execution of smart contracts in a decentralized way. A smart contract is a piece of self-executing computer code written in a high-level language (e.g., Solidity). Once compiled, contract's bytecode is hosted by a blockchain and can be triggered by blockchain transactions to execute some built-in functionality of the contract. Once deployed on the chain, the contracts are immutable, i.e., they cannot be modified or deleted from the chain, which in essence provides transaction integrity verification in trust-less environment.

Ethereum [3] was the first platform to support smart contracts, with many more to follow (e.g., Tezos [4], EOS [5], Cardano [6], and Hyperledger Fabric [7], Lisk [8]).

The rapid adoption of cryptocurrencies quickly escalated the severity and frequency of malicious activities on the blockchain. To address this problem, a significant amount of research was dedicated to detection of vulnerabilities in smart contracts before their deployment on the blockchain [9–23]. Focusing on the contract's code, these approaches fail to detect illicit activity on the chain after the contract deployment.

To address this issue, research studies focused on detection of either generic malicious transactions [24–26] or known illicit activities. For example, honeypot smart contracts [27], phishing attacks [28–32], and Ponzi scheme activities [33–37]. The overwhelming majority of these research studies focus solely on identifying a specific type of malicious behavior, thus limiting their applicability in real environment.

A more few solutions are capable of detecting suspicious transactions though instrumentation of Ethereum nodes or contracts [38–41]. While these approaches are capable of detecting suspicious accounts early, they require modification of an Ethereum client and tend to incur significant resource consumption, and as a result are mostly not suitable for a large scale detection. Studies that do not leverage instrumentation require replaying historic transactions similarly incurring a significant cost (e.g., HORUS [42], EthScope [43]).

In spite of over a decade of research, the lack of practical tools still presents a significant hindrance for the early detection of malicious activities on the blockchain. In practice, platforms often resort to manual analysis and labelling of suspicious activity.

To tackle this challenge, in this article, we introduce EtherShield, a novel time-interval-based incremental analysis for detection of malicious activity on Ethereum blockchain. As opposed to the existing approaches, EtherShield considers the natural evolution of the real-time Ethereum chain's state. As different malicious activity is manifested through different characteristics, in our analysis, we assess accounts through transaction activity and code-based patterns. These two complementary perspectives allow us to reveal hidden illicit behavior. For example, it is evident from existing studies [28] that communication between accounts can characterize phishing attacks, yet, an analysis of phishing contracts' code often provides limited evidence of their maliciousness.

To allow for expedited detection of illicit behavior, EtherShield employs a time-interval-based analysis. Unlike the existing methods that depend on complete transaction history for the analyzed accounts, we leverage temporal transaction information to provide an immediate insight into contracts' behavior. We rely on time intervals to provide us with a historical record, allowing us to identify malicious activity based on the amount of available information. This strategy is more feasible for deployment in real-life scenarios, as it is often impractical to delay detection until all transaction data becomes accessible.

EtherShield utilizes an incremental approach to address situations where the accuracy of the detection result is insufficient for the current time frame. Time-interval-based analysis enables EtherShield to incrementally repeat classification with larger time intervals to improve detection accuracy or until conclusive detection results are achieved. It is important to note that this

incremental approach does not necessitate the presence of all available data, thus enabling immediate detection while progressively improving detection accuracy as more data becomes available.

We validate the utility of EtherShield on a set of 15,264 Ethereum accounts among which 5,320 are benign and 9,944 represent five different types of malicious activities. In our evaluations, EtherShield is able to detect specific type of malicious activity with 86.52% detection accuracy using 1-hour interval data and 91.33% accuracy with 1-year data. Compared to the existing approaches that commonly leverage significant transaction history, EtherShield provides an expedited detection, achieving comparable accuracy with significantly less data than what is required by other approaches.

Our main contributions can be summarized as follows:

— We design and implement EtherShield, a new time-interval-based incremental analysis for detection of different types of malicious activity on Ethereum blockchain.
— We evaluate the effectiveness of EtherShield and investigate the accuracy of detection over 15 time intervals.
— We compare EtherShield against state-of-the art blockchain scam detection approach.

To facilitate research in this field, we made the code and datasets used in this study publicly available.[1]

The rest of this work is organized as follows: Section 2 gives an overview of the related work in the field, Section 3 illustrates the background of Ethereum. Section 4 introduces our approach for solving the security problem on Ethereum. Section 5 gives the details for experimental setup, and Section 6 gives a detailed description of the our experiments for our approach. Section 7 analyzes the experiment results, and Section 8 is the conclusion of this study.

2 RELATED WORK

Continued adoption of cryptocurrencies escalated the malicious activities on the blockchains, attracting significant research attention. The earlier studies in the field primarily aimed at detection of vulnerabilities in smart contracts through static analysis of code. Most of these studies rely on static analysis of the code in an offline setting, i.e., analysis of code without its execution (e.g., Vandal [9], Slither [10], Oyente [11], Securify [12], Solc-Verify [50], and Mythril [51]). Mostly based on the patterns of known insecure behavior in code, these studies view contracts in isolation, hence, failing to take into account dynamic contract interactions occurring on the chain after contract's deployment. Our approach is complementary to these static analysis tools, as it aims to detect all suspicious accounts not limited to those containing contract's code.

More recently, the research has shifted to a general detection of malicious activity that involves contracts already deployed on the blockchain. We broadly divide these studies based on the type of features they leverage for their analysis (Table 1).

Code-based analysis. The significant number of the existing research studies for online security analysis of smart contracts analyze a contract's execution flow using extensive instrumentation of an Ethereum client (e.g., SODA [17], ECFChecker [21], Sereum [22], TXSpector [40], EtherProv [41], and ContractGuard [39]. While this approach provides fine-grained runtime information, it incurs significant resource consumption and requires modification of an Ethereum client, consequently making this approach mostly not suitable for a large-scale detection.

Studies that do not leverage instrumentation require replying historic transactions similarly incurring a significant cost (e.g., HORUS [42], EthScope [43]).

The vast majority of this research leverages patterns of known vulnerabilities. Along these studies, researchers use machine learning classification for identification of suspicious contracts.

[1] https://cyberlab.usask.ca/ethershield.html

Table 1. The Overview of the Existing Studies Focusing on Detection of Malicious Activity on Ethereum Blockchain

Related Work	Year	Detection focus	Approach	Features Type
Chen et al. [33]	2018	Ponzi	XGBoost	Transaction Activity & Code Features
Steven et al. [24]	2020	Illicit	XGBoost	Transaction Activity Features
Chen et al. [28]	2020	Phishing	GCN	Transaction Activity Features
Lou et al. [34]	2020	Ponzi	CNN	Code Features
HoneyBadger [44]	2020	Honeypot contracts	XGBoost	Code Features
SADPonzi [45]	2021	Ponzi scam	XGBoost	Code Features
Li et al. [30]	2021	Phishing	GNN	Transaction Activity Features
Zhang et al. [35]	2021	Ponzi	CatBoost	Transaction Activity & Code Features
Aljofey et al. [36]	2021	Ponzi	KNN, DT, AdaBoost, RF, ET, GB, XGBoost	Code Features
Wen et al. [31]	2021	Phishing	SVM, KNN, AdaBoost	Transaction Activity Features
Teng et al. [25]	2021	High risk	XGBoost, LSTM, GRU, RF	Transaction Activity Features
Hara et al. [27]	2021	Honeypot contracts	XGBoost	Transaction Activity & Code Features
Wang et al. [46]	2021	Phishing	Graph Embedding	Transaction Activity Features
Zhang et al. [37]	2022	Ponzi	LightGBM	Transaction Activity & Code Features
Xia et al. [32]	2022	Phishing	Graph Embedding, XGBoost	Transaction Activity Features
Hou et al. [47]	2022	Phishing	GCN, CRF	Transaction Activity Features
Luo et al. [48]	2023	Phishing	Graph Embedding	Transaction Activity Features
Li et al. [49]	2023	Phishing	Deep Learning	Transaction Activity Features

Among these are the approaches for detecting Ethereum smart contracts involved in Ponzi schemes [33, 35–37, 45, 52]. As the Ethereum chain only retains smart contracts' bytecode, these studies disassemble bytecode to extract opcodes/operands n-gram features that are then used in classification analysis. Similar approaches were introduced for detection of honeypot smart contracts that aim to fraud users by leveraging hidden traps in the contract code [27, 44]. Camino et al. introduced HoneyBadger, a symbolic execution-based technique that leveraged handcrafted rules to discover honeypot techniques in the contract bytecode [44]. Hara et al. [27] also only relied on the smart code features or detection. The approach employed the **term-frequency inverse document-frequency (TF-IDF)** method to extract smart code features and word2vec embedding to obtain the distributed representations of contract opcodes.

Transaction activity-based analysis. Most of the existing security analysis studies are designed to analyze smart contract bytecode. A transaction-based approach has received a relatively less attention.

As blockchain transactions naturally form a graph, most of these transaction-based detection approaches leverage graph-based analysis. For example, observing the difference in the number of incoming transactions for phishing accounts, Wen et al. [31] proposed to analyze transactions of phishing accounts and their first-order neighbors to detect instances of phishing attacks in Ethereum. Similar approaches based on the analysis of the adjacency relationship through transactions for detection of phishing were developed by other researchers [28–30, 32, 46]. Chen et al. [28] detected phishing addresses by treating accounts and transactions as nodes and edges in a **Graph Convolutional Network (GCN)**. Lou et al. combined **Convolutional Neural Network (CNN)** with bytecode of smart contracts to detect Ponzi addresses [34]. Hou et al. [47] leveraged GCN and **Conditional Random Field (CRF)**, while Li et al. [49] used deep learning approach for phishing accounts detection. The developed graph-based approach called SIEGE in addition to traditionally

considered attributes of nodes (accounts) combined in the graph the contexts in which the nodes are located in the graph. This perspective allowed to predict the future account behavior given context.

Xia et al. [32] developed an attributed ego-graph embedding framework to represent Ethereum transactions of phishing accounts. Learning graph embeddings for each of these ego-graphs using Graph2vec [53] gives a set of features that are then used in machine learning classification to distinguish phishing accounts. Later, Luo et al. [48] proposed a new random walk-based network embedding algorithm called bias2vec to improve the detection accuracy. To offer an analysis of patterns formed by interaction of transactions targeting phishing accounts, Wang et al. [46] introduced **transaction subgraph network (TSGN)** model. TSGN utilizes the original raw transaction information to produce a representation in the form of a graph, where individual accounts are depicted as nodes and the transactions between them are shown as edges. The edges are annotated with additional information, among which is the edge weight that denotes the quantity of cryptocurrencies transferred by transactions. TSGN is the parsed to extract necessary graphs-based features for classification using Random Forest classifier.

None of these graph-based methods aims for detection of multiple threats and are typically geared towards particular type of malicious activity, which is in most cases phishing.

More generic models were considered by Farrugia et al. [24]. As opposed to the graph-based analysis, Farrugia et al. developed a model for differentiating between benign and malicious accounts based on 42 features describing accounts and the corresponding transactions. This is the most closely related approach to our study.

Among the studies that investigated multi-classification for Ethereum smart contracts are Hu et al. [25] and Shi et al. [26]. Both studies investigated broad contracts' transaction behavior with an objective to understand common patterns among contracts and affiliate them with high-level categories such as gaming, gambling, social, finance, and so on. Among these categories, Hu et al. [25] also considered high-risk smart contracts for their potential to contain code vulnerabilities. Although being overly broad in their analysis, both studies narrowed their focus in experiments considering only a few basic account features (e.g., number of transactions, balance).

3 BACKGROUND

3.1 Ethereum

Proposed in 2013, Ethereum is a public blockchain that enables developers to create, deploy, and execute programs on the Ethereum network. These programs, called smart contracts, are written in high-level programming languages such as Solidity.

To deploy a contract on the Ethereum chain, a user compiles a contract to the **Ethereum Virtual Machine (EVM)** bytecode and then deploys its compiled bytecode by issuing a transaction, i.e., cryptographically signed instruction from his account. Once published on the blockchain, contracts are immutable. The overview of this process is shown in Figure 1.

To execute a smart contract located on the chain, a user can initiate a transaction to an address that contains the smart contract. Transactions require validation prior to publishing on the chain, thus users are required to pay a "gas" fee for this computation. Gas is paid in Ethereum cryptocurrency, called *ether*. In addition to this, each account has a balance in Ether that can be modified by sending it some ether.

3.2 Ethereum Accounts

To support pseudo-anonymity, users on the blockchain are identified by unique accounts. There are two types of accounts on Ethereum: ***Externally owned account (EOA)*** and *contract account*. Smart contract account is an account associated with a public contract that resides on a chain.

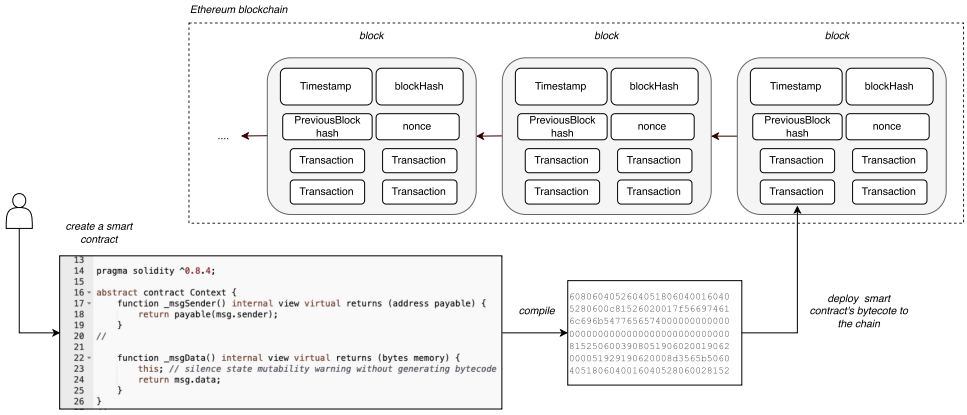

Fig. 1. An overview of the Ethereum blockchain.

The address of a smart contract account is derived from the owner's address. However, EOAs are managed by users, thus the address of an EOA is determined by the user's public key.

A user represented by an EOA can initiate a transaction that in turn can execute a smart contact published on the chain. These transactions are known as *normal transactions*. Contract accounts cannot initiate new transactions on their own, but can issue transactions in response to other transactions they have received. For example, smart contracts may interact with each other by invoking the corresponding functions in other contracts. These kinds of transactions are called *internal transactions*.

3.3 Ethereum Standards

Ethereum is a powerful platform that is leveraged for a variety of services. Among these services is the exchange of tokens that can represent various functionality. To help facilitate the interactions between different tokens, several standards were developed (e.g., ERC20, ERC721, ERC1155). Among these standards, ERC20 is the most commonly used interface standard. Hence, smart contracts implementing the interactions among tokens have to follow this standard. Any transactions related to the transfer of tokens under a given interface standard are labeled as such, e.g., transactions on the ERC20 interface standard are referred to as *ERC20 transactions*.

4 THE PROPOSED APPROACH

One of the significant obstacles in prompt detection of malicious activity on Ethereum is the availability of context required to identify it. For example, phishing scams are not immediately apparent on the chain, and therefore their identification is typically triggered by users' reports indicating suspicious behavior, consequently leading to delayed detection. The delay in detection presents a more significant problem in blockchain than in any other domain due to the inherit immutability of accounts. Without a possibility to remove malicious accounts, the only viable option is to alert users of the blockchain platform and label the corresponding accounts as malicious.

Our objective is to offer an expedited detection even when data is limited. To address this, EtherShield relies on the training and detection stages presented in Figure 2.

The training component relies on a set of accounts known to be abnormal or malicious and labelled either manually or automatically. The accounts are augmented with the corresponding smart contracts and historical transaction information extracted from the blockchain. The contracts and transactions are then passed to the parsing module that extracts feature vectors

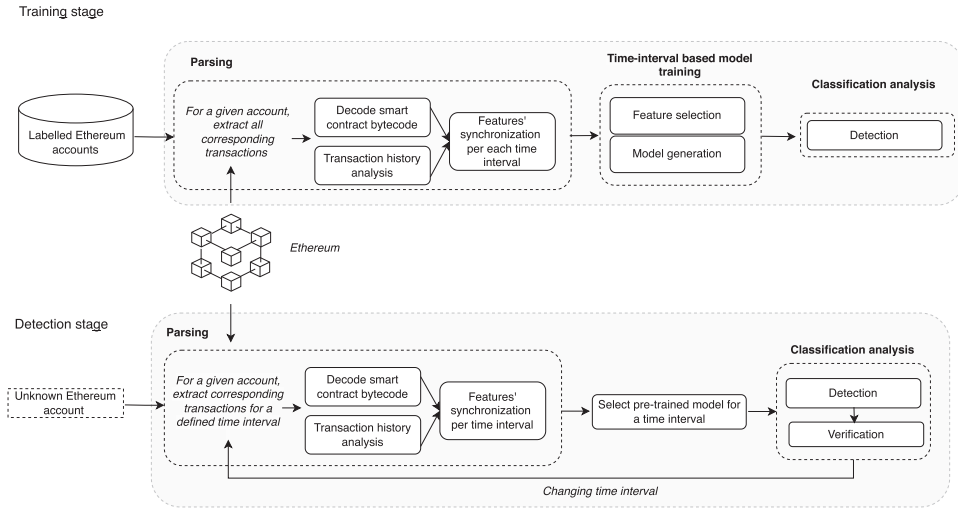

Fig. 2. The workflow of EtherShield.

from a disassembled smart contract's bytecode and time-interval-based analysis of transactions' data. For a given contract, the decoding process decodes the bytecode to opcodes and operands. Ethereum's smart contracts and the underlying bytecode are immutable, hence, once extracted, opcodes and operands of a smart contract remain the same. Conversely, transaction information is subject to change over time intervals, leading to varying transaction-related feature vectors that reflect the specific time intervals in consideration. The resulting feature vectors are synchronized to characterize a smart contract behavior per time interval. Since each time interval embeds a different amount of information, the training module generates classification models per time interval.

In the **detection stage**, we are provided with a new account to determine whether it exhibits any malicious behavior. For this account, we retrieve the available transactions that occurred during a defined time interval. As opposed to the training stage, the goal is not to extract all historical information but rather information for the time interval that is available (e.g., if contract was recently deployed on the chain) or the most effective for malicious behavior detection (e.g., if a significant time has passed since the publishing of a contract). This helps facilitate rapid detection of malicious behavior. Note that in some cases, e.g., when a significant amount of time has passed, selecting a smaller time interval also means that an original contact may not be retrieved from the chain. Hence, it is critical for detection to be flexible enough to accommodate these scenarios and provide accurate detection based only on transaction activity.

Similarly to the training stage, the parsing module then extracts contract's opcodes and operands (if contact is available), and temporal feature vectors from the corresponding blockchain transactions. The classification model is pulled from the set of generated models, depending on the selected time interval. The classification analysis module then analyzes contracts behavior to detect malicious activities.

EtherShield leverages an *incremental approach* to address cases where detection result is not sufficiently accurate for a current time interval, e.g., classification rate is below a tolerable threshold. In these cases, transaction information for a larger time interval is requested and classification analysis is repeated. This only becomes robust due to time interval-based analysis of the proposed approach. The availability of pre-trained time interval-based models allows to quickly adjust the analysis interval and obtain results with a larger amount of data. Note that incremental approach

does not require presence of all available data and hence allows for immediate detection, while enabling an incremental improvement in detection accuracy as more data becomes available.

4.1 Parsing Module

For a given account addresses, our parsing module retrieves the corresponding transaction history. For each transaction, we retrieve timestamp, sender, and receiver addresses, amount of ether transferred, and data field. If an account is associated with a contract, then we also retrieve contract's bytecode. Our analysis aims to encompass insecure behavior regardless of the contract presence. Hence, we analyze characteristics exhibited both at the code level (if contract is present) and transaction level. Based on the extracted information, the parsing module extracts *transaction activity features* characterizing an account's transaction history and *code features* characterizing the smart contract bytecode deployed on the chain.

4.1.1 Transaction History Analysis. We refer to the activity of an account regardless of its type as a *transaction activity*. To capture malicious activity, we extract characteristics providing statistical and financial context for each account and the status of transactions (if reverted).

Statistical features: To represent account interactions, we aggregate information about outgoing and incoming transactions through basic measurements such as total frequency, time interval between the first and the last transaction, average time between transactions, the longest time between transactions, and ratios between types of transactions. Depending on the transaction type, transaction activity may reflect the characteristics of various attacks. For instance, phishing attack is characterized by multiple normal transactions invoking a contract, followed by one or a few internal transactions to transfer the funds. To fully capture attack characteristics, we calculate statistical features for normal, internal, and ERC20 transactions.

Reverted features: The majority of transactions after a successful validation become a part of the blockchain. A smaller portion of transactions, however, fail this step. These are known as reverted transactions. Although commonly ignored, they comprise an important part of blockchain ecosystem. Reverted transactions serve as a mechanism to prevent smart contracts exhibiting abnormal behavior from being published on the chain [54]. We thus include these transactions in our analysis.

Financial features: To understand how the flow of funds impacts detection of malicious activities, we derive supplemental features describing the amounts of ether accompanying each of the statistical features.

Summary: Overall, we derive 263 features characterizing transaction activity, including 145 statistical, 3 reverted, and 115 financial features, shown in Table 2. Note that, depending on the feature, we calculate a different set of values. For most features, we extract values corresponding to the normal transactions, internal transactions, ERC20 transactions, ERC721 transactions, ERC1155 transactions, and all transactions cumulatively. For some features this is not feasible. For example, at the time of analysis, reverted transactions were only provided by Etherscan platform for normal and internal transactions, thus the corresponding calculation only includes number of reverted normal and internal transactions and all reverted transactions. Other features are intentionally constructed to reflect single types of transactions, e.g., the proportion of ether amount sent by normal transactions.

4.1.2 Decode Smart Contract Bytecode. We augment transaction activity with features derived from the corresponding smart contract bytecode. Since we are dealing with contracts already deployed on the blockchain, only their bytecode is typically available for analysis. We disassemble contract's bytecode and use n-gram approach to tokenize the contract's bytecode and opcodes/operands stream. In our tokenization, we employ $n = 4$, i.e., the bytecode and

Table 2. Transaction Activity Features

Statistical features:	
The average time between incoming transactions (6)	The average time between outcoming transactions (6)
Time since the first until the last transaction (6)	The longest interval between two transactions (6)
The shortest interval between two transactions (6)	The total number of transactions (6)
The total number of incoming transactions (6)	The total number of outcoming transactions (6)
The number of unique incoming addresses (6)	The number of unique outcoming addresses (6)
The proportion of unique incoming address out of all incoming transactions (6)	The proportion of unique outcoming address out of all outcoming transactions (6)
The proportion of normal incoming transactions out of all normal transactions (1)	The proportion of normal outcoming transactions out of all normal transactions (1)
The proportion of normal transactions out of all transactions (1)	The proportion of normal incoming transactions out of all incoming transactions (1)
The proportion of normal incoming transactions out of all transactions (1)	The proportion of normal outcoming transactions out of all outcoming transactions (1)
The proportion of normal outcoming transactions out of all transactions (1)	The proportion of internal transactions out of all transactions (1)
The proportion of internal incoming transactions out of all internal transactions (1)	The proportion of internal outcoming transactions out of all internal transactions (1)
The proportion of internal incoming transactions out of all incoming transactions (1)	The proportion of internal incoming transactions out of all transactions (1)
The proportion of internal outcoming transactions out of all outcoming transactions (1)	The proportion of internal outcoming transactions out of all transactions (1)
The proportion of ERC20 transactions out of all transactions (1)	The proportion of ERC20 incoming transactions out of all ERC20 transactions (1)
The proportion of ERC20 outcoming transactions out of all ERC20 transactions (1)	The proportion of ERC20 incoming transactions out of all incoming transactions (1)
The proportion of ERC20 incoming transactions out of all transactions (1)	The proportion of ERC20 outcoming transactions out of all outcoming transactions (1)
The proportion of ERC20 outcoming transactions out of all transactions (1)	The proportion of ERC721 transactions out of all transactions (1)
The proportion of ERC721 incoming transactions out of all ERC721 transactions (1)	The proportion of ERC721 outcoming transactions out of all ERC721 transactions (1)
The proportion of ERC721 incoming transactions out of all incoming transactions (1)	The proportion of ERC721 incoming transactions out of all transactions (1)
The proportion of ERC721 outcoming transactions out of all outcoming transactions (1)	The proportion of ERC721 outcoming transactions out of all transactions (1)
The proportion of ERC1155 transactions out of all transactions (1)	The proportion of ERC1155 incoming transactions out of all ERC1155 transactions (1)
The proportion of ERC1155 outcoming transactions out of all ERC1155 transactions (1)	The proportion of ERC1155 incoming transactions out of all incoming transactions (1)
The proportion of ERC1155 incoming transactions out of all transactions (1)	The proportion of ERC1155 outcoming transactions out of all outcoming transactions (1)
The proportion of ERC1155 outcoming transactions out of all transactions (1)	The proportion of all incoming transactions out of all transactions (1)
The proportion of all outcoming transactions out of all transactions (1)	The average number of transactions per day (6)
The average number of transactions per hour (6)	The average number of incoming transactions per day (6)
The average number of outcoming transactions per day (6)	The average number of incoming transactions per hour (6)
The average number of outcoming transactions per hour (6)	
Reverted features:	
The number of reverted transactions (3)	
Financial features:	
The minimum amount ever received (6)	The maximum amount ever received (6)
The average amount received for each transaction (6)	The minimum amount ever sent (6)
The maximum amount ever sent (6)	The average amount sent for each transaction (6)
The total amount ever received (6)	The total amount ever sent (6)
The average amount transferred per day (6)	The average amount transferred per hour (6)
The average incoming amount per day (6)	The average outcoming amount per day (6)
The average incoming amount per hour (6)	The average outcoming amount per hour (6)
The total amount transferred by outcoming and incoming transactions (6)	The proportion of amount transferred by normal transactions out of all amount transferred (1)
The proportion of amount sent by normal transactions out of all sent amount (1)	The proportion of amount sent by normal transactions out of all amount transferred (1)
The proportion of amount received by normal transactions out of all received amount (1)	The proportion of amount received by normal transactions out of all amount transferred (1)
The proportion of amount transferred by internal transactions out of all transferred amount (1)	The proportion of amount sent by internal transactions out of all sent amount (1)
The proportion of amount sent by internal transactions out of all amount transferred (1)	The proportion of amount received by internal transactions out of all received amount (1)
The proportion of amount received by internal transactions out of all amount transferred (1)	The proportion of amount transferred by ERC20 transactions out of all amount transferred (1)
The proportion of amount sent by ERC20 transactions out of all sent amount (1)	The proportion of amount sent by ERC20 transactions out of all amount transferred (1)
The proportion of amount received by ERC20 transactions out of all received amount (1)	The proportion of amount received by ERC20 transactions out of all amount transferred (1)
The proportion of amount transferred by ERC721 transactions out of all amount transferred (1)	The proportion of amount sent by ERC721 transactions out of all sent amount (1)
The proportion of amount sent by ERC721 transactions out of all amount transferred (1)	The proportion of amount received by ERC721 transactions out of all received amount (1)
The proportion of amount received by ERC721 transactions out of all amount transferred (1)	The proportion of amount transferred by ERC1155 transactions out of all amount transferred (1)
The proportion of amount sent by ERC1155 transactions out of all sent amount (1)	The proportion of amount sent by ERC1155 transactions out of all amount transferred (1)
The proportion of amount received by ERC1155 transactions out of all received amount (1)	The proportion of amount received by ERC1155 transactions out of all amount (1)

Values in parentheses indicate the number of values calculated for each feature.

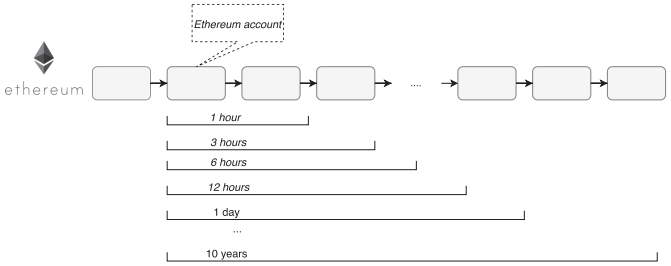

Fig. 3. Time interval-based feature extraction.

opcodes/operands stream are broken into tokens, where each token is the corresponding hex number representing a byte, one opcode, or one operand. (We use 4-gram for parsing bytecode, parsing the combination of opcode and operand codes, but take one opcode and one operand as a single feature.)

To rank the resulting n-grams, we calculate **TF-IDF (term frequency-inverse document frequency)** that evaluates the importance of a token for a given contract as follows:

$$TF - IDF(w, c, C) = \log(1 + f(w, c)) \cdot \log \frac{N}{f(w, C)},$$

where $f(w, c)$ is the frequency of a token w in a contract c and N is the number of contracts in the collection C.

Based on the TF-IDF score, we rank our n-gram features and select the top 50 features for each category to form our feature set.

Our final feature set includes 453 code features, i.e., the overlapped top 50 features for each of six categories (malicious and non-malicious addresses), and smart contract features extracted from bytecode, opcode, and operands. This includes 108 bytecode features (represented as n-grams), 155 opcodes and operands features (represented as n-gram), 123 operand features and 67 opcode features.

4.1.3 Feature Synchronization Based on Time Interval. To capture temporal patterns in transaction activity, we extract features per time interval as shown in Figure 3. The duration of the interval is crucial for temporal patterns. Intervals of different lengths that share the same activity may still reveal different information. For example, the longer period of time is likely to accumulate more transactions revealing malicious behavior, yet shorter periods of time may not have enough transactions to form an anomalous pattern, but are more beneficial to early detection in an event such malicious activity is detected.

In our training stage experiments, we explore a range of time intervals: 1 hour, 3 hours, 6 hours, 12 hours, 1 day, 3 days, 7 days, 14 days, 30 days, 90 days, 180 days, 365 days, 3 years, 5 years, and 10 years. For each of these intervals, we use 263 transaction activity features that cover individual intervals. Thus, the transaction activity features for each time interval along with the corresponding code features form a feature vector for a given interval. We experimentally provide insights into impact of granularity of these time intervals on the accuracy of detection.

During the detection stage, however, the feature vector for a given Ethereum account is formed based on code features (if contract is present), and transaction features calculated for one time interval only.

4.2 Time-interval-based Model Training

The classification model is trained based on each set of selected time intervals. The participating features are selected accordingly, e.g., only features covering 1-hour interval are used in the

training process. This process generates several time-interval-based models that can be used in on-demand classification analysis.

Given a variety of time intervals, we conduct a feature selection analysis. For each time interval, we select the set of best performing features and retain it for further analysis.

4.3 Classification Analysis

The classification module is responsible for analysis of account behavior given a feature vector for a specific time interval.

To guide the accuracy of detection outcome, we employ an additional verification step that can confirm whether achieved detection result is acceptable, e.g., misclassification rate is below a tolerable threshold. If more accurate analysis is required, then transaction information for a larger time interval is requested and classification analysis is repeated with a newly obtained feature vector. For example, if one hour interval proved to be insufficient, then transaction features covering 3-hour interval may be requested and used in the following detection.

Over the past few years, the research on detection of malicious blockchain activity leveraged several classification algorithms in their analysis. In this work, we explore the performance of these classifiers previously employed by other studies: **Random Forest (RF)** [25, 36, 55], **Decision Trees (DTs)** [36], XGBoost [24, 25, 33, 36], and LightGBM [37, 56].

DTs [57] is an interpretable algorithm that produces a sequence of rules to classify the data for a given target. Hence, every decision made by an algorithm can be associated with a corresponding path, which provides a high-level interpretability. With this advantage though the classifier can be unstable when small variations in the data that can result in generation of a completely different tree. It is also known to be prone to overfitting. An RF classifier [58] is an ensemble that fits a number of decision trees on various sub-samples of datasets and uses the average to improve the predictive accuracy of the model.

While RF and DT are popular classifiers, XGBoost and LightGBM have recently seen an increase in popularity within the data science community.

The decision tree algorithms (including models such as XGBoost and LightGBM) integrate different hypotheses in a single model and thus provide better performance than decision tree-based classifiers.

XGBoost [59] is a decision-tree-based algorithm that utilizes the gradient boosting architecture. Gradient boosting is a machine learning ensemble method that involves combining multiple weak models to create a strong predictive model. The architecture of a gradient boosting model typically involves creating decision trees as the weak learners and then combining them using a boosting algorithm. XGBoost is an ensemble machine learning method that supports three types of gradient boosting: gradient boosting, stochastic gradient boosting, and regularized gradient boosting, which provides a faster classification. XGBoost supports auto pruning, which stops the growth of the decision tree to help reduce overfitting and improve accuracy.

LightGBM [60] is a novel **Gradient Boosting Decision Tree algorithm (GBDT)** that integrates results of multiple weak learners to improve classification result. As such, the model is resistant to overfitting and results are often better than with a single weak learner.

5 EXPERIMENTAL SETUP

5.1 Data

For our analysis, we have collected data representing five malicious categories: phishing, Ponzi schemes, coin laundering, honeypot contracts, and high-risk contracts associated with known code vulnerabilities, such as reentrancy bugs, arithmetic overflow, unchecked low-level calls (for details, see Reference [61]).

Table 3. Dataset

Category	Total accounts	Smart Contract accounts	EOA accounts
Phishing	5,186	215	4,971
Ponzi	298	283	15
Coin laundering	826	1	825
Honeypot	1,162	1,070	92
High Risk	2,472	2,421	51
Benign	5,320	391	4,929
Total	15,264	4,381	10,883

Table 4. The Parameters of the Classification Algorithms

Alg.	**Hyperparameters**	**Classifier**
RF	criterion='gini', max_features='sqrt', min_samples_leaf=1, min_samples_split=2, n_estimators=115	Nonlinear
DT	criterion='log_loss', max_depth=13, min_samples_leaf=3, min_samples_split=17, splitter='best'	Nonlinear
XG	gamma=0, max_delta_step=0, max_depth=7, min_child_weight=5	Nonlinear
LGBM	max_delta_step=8, max_depth=8, num_leaves=21	Nonlinear

Phishing contracts and contracts associated with coin laundering, i.e., contracts performing activities with stolen coins, were retrieved from Etherscan.io, an analytics platform for Ethereum. Etherscan's labelling of malicious smart contracts is broad and imprecise. We thus extracted all accounts labelled as a "Phish/Hack" and through manual analysis filtered out those that included other tags, leaving only phishing tags. For coin laundering, we collected accounts labelled a "Bitpoint Hack," a "Cryptopia Hack," a "EtherDelta Hack," a "Lendf.Me Hack," and a "Upbit Hack," resulting in 826 addresses.

We obtained a dataset with 184 Ponzi addresses used in Massimo et al.'s study [62] and augmented it with the additional samples used by Chen et al. [38] and He at al. [63] studies. After de-duplication, our final set contained 298 Ethereum addresses associated with Ponzi scheme activities. We used a public project collected by the HoneyBadger Project [64] with additional 24 samples obtained from Torres et al.'s study [65].

The high-risk samples were selected from the dataset provided by Liao et al. [61]. We have randomly selected 2,472 addresses, 2,421 of which were associated with smart contracts accounts. In our analysis, we have used a set of 5,320 benign addresses employed by Kumar et al. [66].

The overview of our dataset is shown in Table 3. Since our analysis is based on the transaction history of each address, we leveraged the Etherscan.io API [67] to extract transaction history, contract information, and ether amounts.

5.2 Experimental Parameters

The system was implemented using the Python language (v.3.8.4) with the scikit-learn library (v.0.23.1). We adopted Python pyevmasm library [68] to decode the bytecode to opcodes and operands. A summary of the classification algorithm hyperparameters is given in Table 4. For hyperparameter optimization, we used the grid search approach. Four-fold cross-validation was employed to measure the accuracy of the machine learning model. All experiments were performed on a Ubuntu server equipped with 32 GB of RAM and 10 CPU cores.

6 EXPERIMENTS

We perform several experiments to validate our approach and examine its effectiveness for various time intervals. To estimate accuracy of analysis, we used stratified 4-fold cross-validation. To evaluate our results, we use weighted-average accuracy, *accuracy* in short, defined as a percentage of accounts correctly detected as benign or attributed to the corresponding threats over the total number of accounts, where accuracy of each class is weighted by the number of accounts from that class.

6.1 Baseline Experiment

To provide a baseline for our experiments, we mimic the conventional approaches that typically employ all available for the analysis data. For these experiments, we use all available features without feature selection.

Our dataset includes accounts that exhibited transaction activity for almost 10 years. Figure 6 shows the frequency of accounts with the available transaction history, i.e., an elapsed time between the first and the last transaction for accounts available in our dataset.

The largest amount of 2,952 accounts in our set encompass less than one hour worth of transactions. A significant amount of accounts (2,108 and 1,881) have an active account history spanning 3 years and 5 years. It is interesting to note that the vast majority of accounts (both benign and malicious) become significantly less active after 5 years. Most accounts do not show any activity after 5 years of their appearance on the chain. The longest time span that transactions cover for some accounts is up to 10 years (330 accounts). In our set, no account has been active for 10 or more years. Hence, our baseline, non-time-based analysis uses all transactions (up to 10 years, when available) for all accounts. Note that not all contracts placed on the blockchain are used by the community, i.e., some contracts are never executed by users. In these cases, contracts have no history beyond the initial transaction that places the contract on the chain. Similarly, a small percentage of EOA accounts has no transaction history beyond 1 transaction. To indicate these cases, we use a "no history" label.

The baseline analysis results are presented in Table 5. Since our dataset contains labels indicating a specific threat, for binary classification, we have relabelled the data as malicious or benign regardless of the specific malicious activity type.

Our results with smart contract features confirm the limitations of code-based only analysis. The accuracy does not exceed 67% in binary classification and 61% in multi-class classification. This is consistent with other studies showing better performance of code-based features on contracts containing code vulnerabilities (we have one such category: high risk) and lower accuracy for threats unrelated to code [25, 69].

The accuracy with transaction activity features reaches 90%–93%. However, we see a further improvement in accuracy with the combination of code-based and transaction activity features.

While, in general, binary classification performs well, the goal of our analysis is to differentiate various types of malicious activity.

Among four classification algorithms we use, XGBoost and LightGBM performed the best across all experiments. We therefore conduct the rest of our experiments with these two algorithms.

6.2 Feature Selection Analysis

Our baseline analysis was performed on all 716 (453 code and 263 transaction) features. We conducted feature selection to assess the importance of these features in our analysis. Both XGBoost and LightGBM offer similar metrics to quantify the importance of features in the trained model. Since both algorithms are based on Gradient Boosting Decision Trees, the feature importance is measured by the number of times a feature is used to split the data across all trees [59, 60]. In

Table 5. Classification Results for Baseline Experiments

	RF	DT	XGBoost	LightGBM
Binary classification				
Transaction activity	93.33%	90.23%	94.01%	93.64%
Smart contract features:				
Bytecode	67.22%	66.93%	67.28%	67.25%
Opcodes only	67.15%	66.74%	67.19%	67.17%
Operands only	67.23%	67.05%	67.24%	67.28%
Opcodes&Operands	67.20%	66.80%	67.22%	67.23%
All features	94.06%	91.27%	**95.30%**	**94.93%**
Multi-class classification				
Transaction activity	88.34%	83.98%	90.25%	89.91%
Smart contract features:				
Bytecode	61.00%	59.60%	61.21%	61.29%
Opcodes only	60.64%	58.58%	60.69%	60.78%
Operands only	60.78%	59.31%	60.88%	60.99%
Opcodes&Operands	60.99%	59.76%	61.31%	61.35%
All features	93.17%	89.86%	**94.91%**	**94.81%**

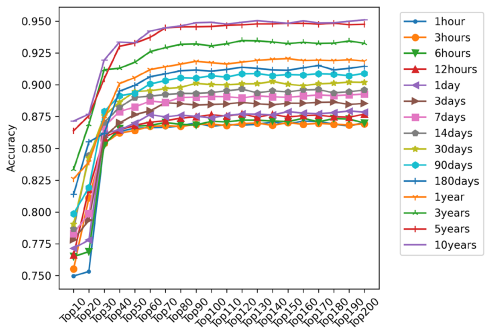

Fig. 4. Feature importance analysis (multi-class classification, XGBoost).

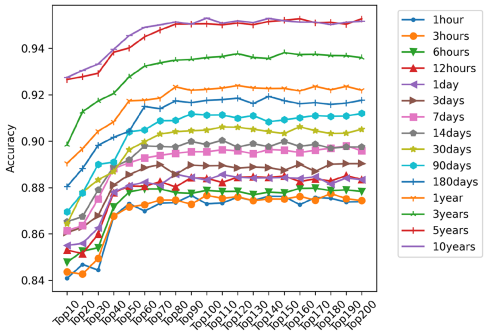

Fig. 5. Feature importance analysis (binary classification, XGBoost).

XGBoost, this measure is referred to as "Weight," while in LightGBM, it is named "Split" [70, 71]. In this study, we evaluate features based on the accuracy loss compared to the baseline analysis performed with all features. For each time interval, we analyze accuracy using the top-ranked subsets of features that are selected based on the feature importance. The accuracy of the models using these selected feature sets are presented in Figures 4 and 5.

As our analysis shows, regardless of time interval, the model performance follows the same pattern. With 20 top-ranked features, both algorithms achieve fairly high and although slightly inconsistent across intervals accuracy (75%–94%). The performance of the models improves with 50 features. The accuracy drastically decreases when smaller sets of features are used. For example, in multi-class classification with XGBoost algorithm, for 5-year time interval, the performance drops from 93.27% (the top 50 features) to 86.38% (the top 10 features). Similarly, in binary analysis, accuracy drops from 94.01% (the top 50 features) to 92.66% (the top 10 features). With the larger feature sets (more than 50 features), the accuracy plateaus show no noticeable improvement.

The close review of the top 50 features sets for different time intervals revealed a significant overlap between sets. To streamline further analysis, we combined the top 50 features across

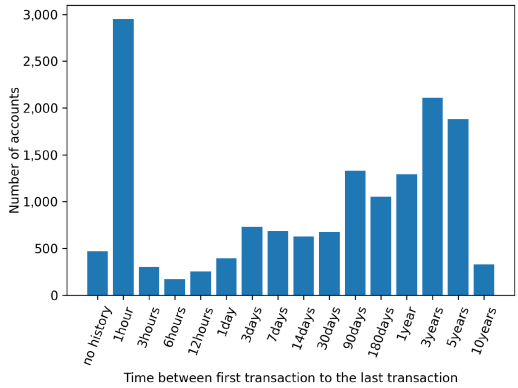

Fig. 6. The available transaction history for the dataset accounts.

different time intervals, deduplicated them, and used the resulting sets for further analysis. Hence, we obtained the top 87 (XGBoost) and 89 (LightGBM) features for multi-class classification, and 89 (XGBoost) and 102 (LightGBM) features for binary classification.

To help analyze the significance of features in the resulting set, we calculated features' frequency as the number of times a feature is used to split the data across all trees (referred to as metrics "Weight" in XGBoost and "Split" in LightGBM). The resulting sets and the corresponding frequency are listed in Tables 8, 9, 10, and 11.

Across these sets, it is evident that transaction activity features dominate these sets, with code features considered less useful in this analysis. For example, sets selected for multi-class classification have 14 (XGBoost) and 31 (LightGBM) code features. Even less code features are found in feature sets generated for binary classification (8 features with XGBoost and 17 features with LightGBM).

It is interesting to note that the top features for both binary and multi-class classification are related to the amount of ether transferred between accounts. Intuitively, this is expected, as malicious actions within the blockchain ecosystem are primarily motivated by financial gains. This intuition is further emphasized by the features characterizing patterns of funds transfer (e.g., interval between transactions, the time since last transaction) appearing among the top 15 features. Hence, the funds transferring patterns serve as one of the primary indicators of malicious behavior.

Regardless of classification type and algorithm, the majority of features across four sets are identical. The following experiments are conducted with the selected top feature sets.

6.3 Time-based Experiments

The existing methods used to detect malicious activity in Ethereum typically depend on having access to the complete transaction history of the accounts being analyzed. Our experiments have validated the high accuracy of this approach. However, in real-life situations, this approach is often impractical, since delaying detection for several years is frequently not feasible.

In this set of experiments, we aim to explore time-based nature of the EtherShield approach. Our analysis is focused on the following time intervals: 1 hour, 3 hours, 6 hours, 12 hours, 1 day, 3 days, 7 days, 14 days, 30 days, 90 days, 180 days, 365 days, 3 years, 5 years, 10 years. Figures 7 and 8 show the classification accuracy across these time intervals.

As expected, accuracy increases as more transaction information becomes available. However, the more practical aspect in this context is how much information is sufficient for reliable detection.

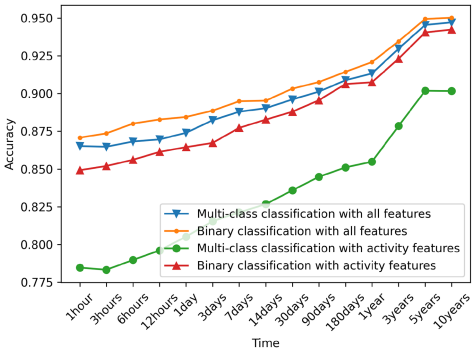

Fig. 7. EtherShield detection results with XGBoost algorithm.

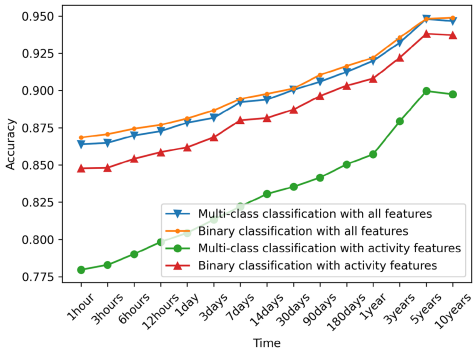

Fig. 8. EtherShield detection results with LightGBM algorithm.

Our results indicate that transactions occurred within 1 hour can be used to predict with 86.84% accuracy whether or not an account is malicious (LightGBM algorithm). We can observe that increasing time interval further improves performance to 89.43% with 7-day time interval and to 92.20% with 1-year interval. This increase is not significant, indicating that the analysis of initial behavior presents a viable option for early detection of malicious activity. Note that the performance of both LightGBM and XGBoost algorithms in these cases is similar.

The detection of specific threats follows a similar pattern. Multi-class classification achieves 86.39% on 1-hour interval data, 89.22% with 7-day time interval, and 91.99% with 1-year interval (LightGBM algorithm). The best accuracy of 95.02% in binary and 94.71% multi-class classification is obtained with 10-year time interval, i.e., all available data for the analyzed accounts (XGBoost algorithm).

This observation establishes the basis for our incremental approach to offer a robust and accurate detection. If a smaller time interval appears to be insufficient, then transaction data features covering the next time interval are requested and classification analysis is repeated with a feature vector covering a larger interval.

For comparison purposes, we have also conducted experiments with transaction activity features only. The baseline experiments showed that transaction features are able to provide a lower (1%–5%), yet are still comparable to all features accuracy.

However, the time-based analysis revealed that the transaction features alone are not sufficient to provide a reliable detection. For example, with multi-class classification, we were not able to achieve even 80% on 1-hour time-interval data, compared to 86.39% accuracy obtained with all features (LightGBM algorithm).

6.4 Analysis of Time-interval Generalization

We showed the efficiency of an incremental approach to boost detection accuracy by gradually employing larger time intervals.

Pre-training models for different time intervals in practice, however, requires presence of data for all intervals. Since we advocate for incremental approach, we assume an ability of approach to train models as data becomes available on the blockchain. Yet, training models for larger time intervals requires time. Hence, it may be practical in real-life deployment to consider situations when models pre-trained on smaller time intervals are leveraged in analysis of, for instance, 3- or 6-hour transaction data, while models for larger time intervals are being trained. This allows to avoid a delay in detection.

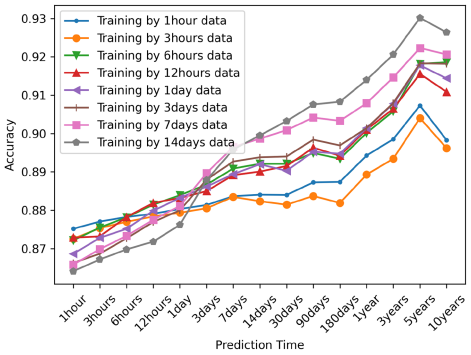

Fig. 9. Binary detection results for models trained with smaller time intervals.

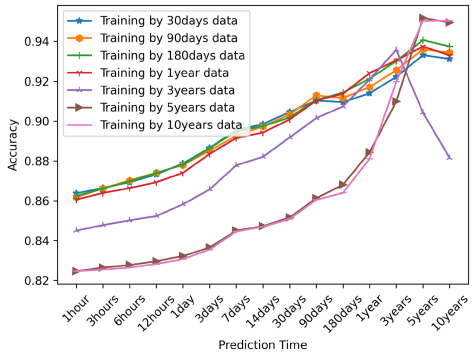

Fig. 10. Binary detection results for models trained with larger time intervals.

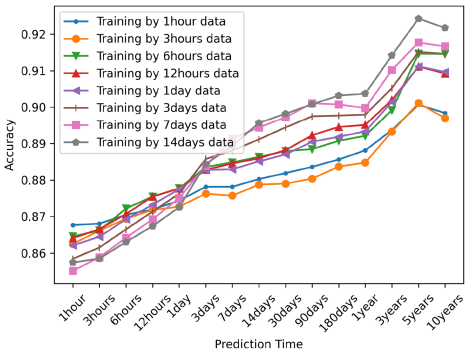

Fig. 11. Multi-class detection results for models trained with smaller time intervals.

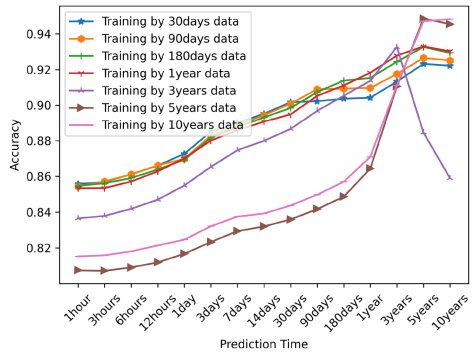

Fig. 12. Multi-class detection results for models trained with larger time intervals.

Hence, one of the key questions in this context is the ability of a classifier model trained on smaller time intervals to generalize, i.e., to handle new scenarios, beyond the time interval used in the model training.

We evaluate the performance of EtherShield for this scenario. The results of model's performance trained on different time intervals with XGBoost are shown in Figures 9, 10, 11 and 12. In these experiments, transaction history for a given interval was used for training the classification model, while the rest of available transaction information was left for testing. For example, a model trained on transaction history accumulated over 1 year was tested on features calculated over 1 hour, 3 hours, 6 hours, and so on, time intervals. Note that this is appropriate in our case, as the features are cumulative, i.e., transaction history accumulated over 1 year incorporates features calculated on all transactions collected for 1-year time interval. Implicitly, this set also includes individual transactions that occurred in the first hour of account placed on the chain. However, features calculated for 1-hour time interval similarly include only cumulative information on all transactions that occurred during this hour. When the testing interval coincided with the training interval, we have only performed cross-validation for a given interval data.

As the results show, the least variability in detection performance is exhibited by model trained on 1-hour data for multi-class classification and 3-hour data for binary classification, i.e., from 86.77% with 1-hour interval to 90.07% accuracy with 3 years (multi-class classification), and in binary analysis, from 87.25% accuracy with 1-hour interval to 90.40% with 5 years. Most models

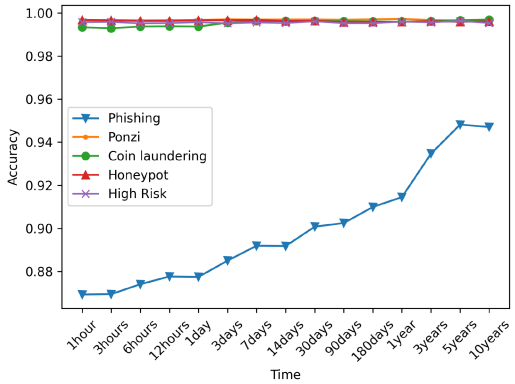

Fig. 13. Analysis of EtherShield performance for various malicious categories.

show a various decrease in accuracy with transaction history of over 3 years. This mostly due to the lack of data as, for example, analysis on 10-year time interval only covers 330 accounts compared to 2,108 accounts with 3-year history.

Minimizing the time interval necessary for training the model is desirable in practice. Among the trained models, most are able to consistently achieve the accuracy above 86%. The difference in performance between the models was in most cases negligible.

Evaluating the length of time interval, several models stood out. For example, models trained on data spanning 7- and 14-day time intervals and similarly, models trained on 1-hour, 3-hours, 6-hours, 12-hours, and 1-day interval data exhibited minor (<0.05%) difference in performance in binary classification. In multi-class analysis, 12-hour model showed a slightly better overall performance.

The practical implication of these experiments is a model trained with a short transaction history can still be effective in accurately predicting account behavior for an extended period (up to 5 or 10 years) when data is limited.

6.5 Analysis of EtherShield Performance for Various Malicious Categories

Our baseline experiments showed the effectiveness of our approach in differentiating malicious from benign accounts. We also demonstrated EtherShield's efficiency in identifying multiple malicious categories against benign accounts. The remaining aspect that we explore in this set of experiments is whether our approach is more (or less) effective in detection of particular type of malicious activity. To investigate this, we perform classification for each category of malicious accounts separately, combining all remaining malicious accounts into a unified category labelled as "others." For these experiments, we employ the XGBoost algorithm. The resulting performance is detailed in Figure 13.

In the context of classifying each individual malicious category, our approach demonstrates a consistently high accuracy. For four malicious categories, we are able to achieve accuracy rates exceeding 99% with just 1 hour's data (specifically, 99.67% for Ponzi, 99.34% for Coin laundering, 99.68% for Honeypot, and 99.57% for High Risk). Subsequently, with more data, the accuracy remains at a relatively high level with over 99% for these four categories. The only notable exception is the phishing accounts. Although our approach can still quite accurately determine phishing activity (86.92% with 1 hour's data and 94.71% with 10-year time period), we observe less consistency. We suspect it is related to the nature of the malicious activity that reveals itself only in communication between accounts, which naturally becomes more evident over time.

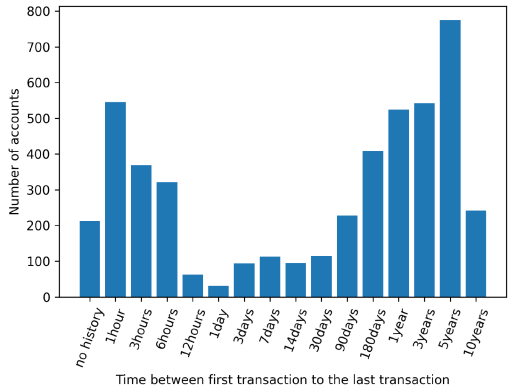

Fig. 14. The available transaction history for Farrugia et al.'s [24] dataset.

6.6 Comparative Analysis

To better assess the proposed approach, we compare the performance of EtherShield with two techniques: detection of malicious accounts in Ethereum proposed by Farrugia et al. [24], and TSGN, the Ethereum phishing classification approach developed by Wang et al. [46].

Our research introduces a novel approach involving a combination of multi-class classification and time-based analysis, which lacks prior comparable methodologies. Consequently, to facilitate meaningful comparisons and establish performance benchmarks, we incorporated several contemporary models that have demonstrated effectiveness in related studies and exhibit the closest resemblance to our approach.

Farrugia et al.'s approach [24] addresses the identification of multiple types of malicious accounts, making it the most closely aligned model. However, TSGN [46] achieves accurate results on their own dataset. TSGN model employs several distinct feature sets, thereby contributing to a more comprehensive representation of existing approaches in the field.

While both studies exclusively focus on binary classification, our research aims to investigate the performance of their models in the context of binary and multi-class classification. We implemented both approaches based on partially available code.[2] In the experiments with Farrugia et al.'s approach, we used 4-fold cross-validation. TSGN approach utilizes different graph representation methods to extract features for phishing account identification. In our experiments, we apply the methods used in the original study: Graph2Vec [53], Diffpool [72], U2GNN [73], and Handcrafted Attributes [74], i.e., 10 features that include Number of addresses, Number of transactions, Average degree of transaction subgraph, Percentage of leaf nodes, Network density, and Average neighbor degree, Average clustering coefficient, Largest Eigenvalue of the adjacency matrix, Average betweenness centrality, and Average closeness centrality.

For fair comparison, we obtained an original dataset employed by Farrugia et al.'s study that consists of 4,676 accounts, 2,502 benign and 2,174 malicious.[3]

As the Figure 14 shows, the timespan of transactions available in Farrugia et al.'s dataset is slightly different from what is found in our data. This set has a significantly less number of accounts

[2]The partial code for approach proposed by Farrugia et al. is available at https://github.com/sfarrugia15/Ethereum_Fraud_Detection. Based on the available code, we have completed the implementation of their approach for a proper comparison. The partial code for TSGN approach is available at https://github.com/GalateaWang/TSGN-master. Additional modifications were done in correspondence with the authors.

[3]We found five invalid accounts in their original dataset. Since the Ethereum data did not exist for these accounts, we excluded them from our experiments.

Table 6. Comparative Analysis (XGBoost)

Time interval	Binary classification				Multi-class classification	
	Our data		Dataset [24]		Our data	
	Farrugia et al. [24]	EtherShield	Farrugia et al. [24]	EtherShield	Farrugia et al. [24]	EtherShield
Reported results	–	–	96.3%	–	–	–
1 hour	82.70%	87.07%	82.87%	91.02%	74.69%	86.52%
3 hours	83.70%	87.36%	84.36%	90.68%	75.85%	86.47%
6 hours	84.21%	88.00%	84.68%	91.25%	77.20%	86.82%
12 hours	85.19%	88.28%	86.37%	90.81%	77.97%	86.96%
1 day	85.83%	88.44%	86.48%	90.53%	78.83%	87.40%
3 days	86.84%	88.86%	89.07%	90.27%	80.06%	88.23%
7 days	88.02%	89.49%	90.89%	90.81%	81.05%	88.79%
14 days	88.21%	89.52%	91.96%	91.68%	81.76%	89.03%
30 days	88.51%	90.32%	92.24%	92.62%	82.55%	89.60%
90 days	89.46%	90.76%	93.86%	93.54%	83.35%	90.12%
180 days	90.29%	91.43%	94.82%	94.74%	83.98%	90.87%
1 year	91.38%	92.08%	95.32%	94.89%	85.17%	91.33%
3 years	91.99%	93.46%	95.57%	94.70%	85.98%	92.97%
5 years	92.43%	94.94%	95.98%	95.49%	86.31%	94.53%
10 years	92.38%	95.02%	95.63%	97.07%	86.39%	94.71%

Table 7. Comparative Analysis with TSGN [46]

Approach	Accuracy
TSGN	
Graph2Vec	72.68%
Handcrafted	71.05%
Diffpool	57.14%
U2GNN	70.76%
EtherShield	93.17%

with transactions scanning 1 hour, i.e., 212 accounts compared to 470 that we have in our set. The largest number of accounts (775) has transactions spanning a period of 5 years, while in our set it is 3 years. The majority of accounts in this set have no activity after five years of creation, with only 242 accounts covering the 5-to-10-year range for transaction activity.

The results of comparative analysis given in Table 6 show that our proposed approach achieves higher accuracy in most cases while providing a more fine-grained detection of malicious behavior.

In the original study, Farrugia et al. [24] reported 96.3% accuracy using XGBoost algorithm using all available transactions (i.e., up to 10 years). This result is comparable to accuracy we obtained with binary classification and Farrugia et al.'s data (95.63%) on a 10-year time interval. *Our approach performs better on both datasets for both binary and multi-class classification.* Most notably, our approach achieves a significantly higher accuracy in cases when data is limited, i.e., 91.02% compared to 82.87% obtained by Farrugia et al.'s approach on 1-hour time-interval data (binary analysis), and 86.52% compared to 74.69% with multi-class analysis. Since Farrugia et al.'s original dataset is binary, multi-class analysis on their data is not feasible.

The comparative analysis results with TSGN approach are given in Table 7. The raw dataset used by Wang et al. [46] was not available, hence, our analysis was only performed on our dataset for all available data. Since the TSGN approach leverages graph representation of transactions, the limited amount of transactions for some accounts resulted in very low performance. We therefore only present the results for the whole dataset. In this analysis, EtherShield was tested on XGBoost algorithm.

In case of TSGN, we were similarly able to obtain significantly better results. Even in case of supervised models such as Diffpool in 100 epochs and U2GNN in 50 epochs (we kept the epoch parameter by the program default), the best accuracy (73.71%) is still significantly lower than EtherShield's result of 93.17%. This experiment demonstrates the advantages of our model, which produces better results than existing state-of-the-art approaches.

7 IMPLICATIONS

Determining maliciousness of Ethereum blockchain account activities is a challenging task. Researchers often rely on the off-chain analysis of contract's code to determine source code vulnerabilities in advance to mitigate the potential problems before code deployment on the chain.

Analyzing activity on the chain has also been explored in research. Yet, in practice, detection of malicious activities is still a mostly manual task. For example, EtherScan, a widely used Ethereum blockchain explorer platform, relies on manual analysis by team experts to identify and label malicious accounts,[4] which is a time-consuming and error-prone process.

In this context, EtherShield presents several important practical implications:

— *Malicious nature of accounts can be determined within 1 hour of accounts appearance on the chain.* Our experiments show that EtherShield can predict with 87.07% accuracy whether or not an account is malicious. This provides a significant advantage, allowing for an expedited detection of suspicious activity. While many studies are able to achieve comparable accuracy, their reliance on all available data is infeasible in practice.

— *Detection accuracy can be incrementally improved without a need to delay detection.* The existing approaches rely on a single snapshot of data in time. Since typically all available data is employed in detection, this limits capabilities of the approaches to offer an improved detection. Our experimental results demonstrate that EtherShield can progressively improve detection of malicious accounts by incrementally increasing the required amount of data for analysis. The incremental approach is robust due to the availability of pre-trained for different time intervals models, which enables quick adjustments to the analysis interval. With this approach, detection accuracy can be improved without having to delay detection until more data becomes available.

— *EtherShield trained with a transaction history spanning a short time interval can still be effective in accurately determining the account behavior for a longer time period.* We show that EtherShield trained on data spanning 3- hour, 12-hour, 7-day, and 14-day time intervals can equally accurately differentiate various types of malicious activity. While accuracy increases as more transaction information becomes available, it is important to have an option to rely on shorter time intervals when data is limited.

— *EtherShield produces better results than existing state-of-the-art approaches.* Our comparative analysis shows that EtherShield achieves consistently better results than other approaches on different datasets for both binary and multi-class classification, while providing a fine-grained detection of malicious behavior types.

8 CONCLUSION

The unprecedented increase in number and diversity of malicious activities in Ethereum blockchain has raised many concerns about the security of the platform, prompting developers and researchers to explore new solutions to mitigate these threats. While a number of techniques were proposed, their limited ability to differentiate threats and reliance on presence of all available transactions limits their practical applicability.

[4]Personal correspondence.

In this article, we introduced EtherShield, a novel time interval-based detection of malicious accounts in Ethereum blockchain. Our analysis showed that EtherShield is effective in detecting different types of malicious behavior even with limited (e.g., up to 1 hour) transaction history. The incremental nature of the approach enables improvements in detection accuracy without a need to postpone detection when more data becomes available.

The proposed approach outperforms the state-of-the-art techniques, offering better accuracy in detection with less amount of data necessary for analysis. The proposed time-interval-based approach presents significant advantages in practical detection, allowing to maintain detection accuracy while significantly reducing the amount of transaction data necessary to determine the malicious nature of activity.

APPENDIX

Table 8. The Top Features for Multi-class Classification (XGBoost)

No.	Features	Frequency	No.	Features	Frequency
1	The minimum amount ever received (Normal transactions)	5,426	2	The minimum amount ever received (All transactions)	4,884
3	The average amount received for each transaction (Normal transactions)	3,453	4	The proportion of amount sent by normal transactions out of all amount transferred	3,143
5	The shortest interval between two transactions (Normal transactions)	3,065	6	The average time between outcoming transactions (Normal transactions)	2,705
7	Time since the first until the last transaction (Normal transactions)	2,679	8	The average incoming amount per day (Normal transactions)	2,627
9	The shortest interval between two transactions (All transactions)	2,512	10	The average amount received for each transaction (All transactions)	2,437
11	The average time between incoming transactions (All transactions)	2,413	12	The minimum amount ever sent (Normal transactions)	2,346
13	The average time between incoming transactions (Normal transactions)	2,247	14	The average incoming amount per day (All transactions)	2,245
15	The longest interval between two transactions (Normal transactions)	2,173	16	The longest interval between two transactions (All transactions)	2,138
17	The proportion of amount received by normal transactions out of all amount transferred	2,130	18	Time since the first until the last transaction (All transactions)	2,028
19	The total amount transferred by outcoming and incoming transactions (All transactions)	1,811	20	The total amount ever received (Normal transactions)	1,591
21	JUMPDEST CALLVALUE ISZERO PUSH2 (opcodes and operands)	1,531	22	The total amount transferred by outcoming and incoming transactions (Normal transactions)	1,525
23	The average amount sent for each transaction (Normal transactions)	1,446	24	The minimum amount ever sent (All transactions)	1,378
25	The total amount ever received (All transactions)	1,316	26	The total number of transactions (ERC20 transactions)	1,198
27	The minimum amount ever received (ERC20 transactions)	1,149	28	0x73ff (bytecode)	1,122
29	The total number of transactions (Normal transactions)	1,098	30	0x60 (operands)	953
31	The average amount sent for each transaction (All transactions)	897	32	The total number of transactions (Internal transactions)	893
33	The shortest interval between two transactions (ERC20 transactions)	815	34	The average time between outcoming transactions (All transactions)	784
35	push1 (opcode)	751	36	The average time between incoming transactions (ERC20 transactions)	655
37	The proportion of normal outcoming transactions out of all normal transactions	635	38	The total number of incoming transactions (Normal transactions)	632
39	The proportion of unique incoming address out of all incoming address (Normal transactions)	566	40	The proportion of normal outcoming transactions out of all transactions	558
41	The proportion of normal incoming transactions out of all transactions	528	42	The average number of incoming transactions per day (Normal transactions)	502
43	The average time between outcoming transactions (ERC20 transactions)	497	44	The average amount transferred per day (Normal transactions)	476
45	The total amount ever sent (Normal transactions)	439	46	The proportion of all outcoming transactions out of all transactions	394
47	The average amount received for each transaction (ERC20 transaction)	382	48	The proportion of unique incoming address out of all incoming transactions (All transactions)	344
49	The proportion of amount received by ERC20 transactions out of all amount transferred	327	50	The total amount ever sent (All transactions)	323
51	0x2 (operands)	312	52	calldatasize (opcode)	304
53	The average incoming amount per day (ERC20 transactions)	290	54	The total number of transactions (All transactions)	286
55	The average number of incoming transactions per day (All transactions)	275	56	SWAP1 DUP2 MSTORE PUSH1 (opcodes and operands)	268
57	0x8fc (operands)	265	58	The longest interval between two transactions (ERC20 transactions)	251
59	The average number of outcoming transactions per day (Normal transactions)	246	60	The average number of transactions per day (Normal transactions)	220
61	The average number of transactions per day (All transactions)	208	62	SWAP1 DUP2 MSTORE PUSH1 (opcodes and operands)	197
63	The average number of outcoming transactions per day (All transactions)	197	64	The proportion of normal transactions out of all transactions	171
65	The proportion of ERC20 incoming transactions out of all ERC20 transactions	153	66	The total amount transferred by outcoming and incoming transactions (ERC20 transactions)	150
67	0xfff1 (bytecode)	144	68	calldataload (opcode)	131
69	The proportion of amount sent by ERC20 transactions out of all amount transferred	131	70	The average amount transferred per day (All transactions)	111
71	Time since the first until the last transaction (ERC20 transactions)	109	72	The number of unique outcoming addresses (Normal transactions)	85
73	The average incoming amount per hour (Normal transactions)	73	74	The proportion of ERC20 incoming transactions out of all transactions	72
75	push8 (opcode)	60	76	0xa0 (operands)	59
77	The average number of outcoming transactions per day (ERC20 transactions)	51	78	The average outcoming amount per day (Normal transactions)	48
79	The total number of incoming transactions (All transactions)	45	80	The proportion of ERC20 transactions out of all transactions	44
81	The minimum amount ever sent (ERC20 transactions)	43	82	The average outcoming amount per day (All transactions)	40
83	The proportion of normal incoming transactions out of all normal transactions	40	84	The proportion of ERC20 outcoming transactions out of all transactions	35
85	The proportion of amount transferred by normal transactions out of all amount transferred	33	86	The total amount ever received (ERC20 transactions)	31
87	0x3 (operands)	26			

Table 9. The Top Features for Binary Classification (XGBoost)

No.	Features	Frequency	No.	Features	Frequency
1	The minimum amount ever received (All transactions)	1,465	2	The minimum amount ever received (Normal transactions)	1,440
3	The proportion of amount sent by normal transactions out of all amount transferred	1,057	4	The average amount received for each transaction (Normal transactions)	900
5	The average time between outcoming transactions (Normal transactions)	890	6	The shortest interval between two transactions (Normal transactions)	866
7	Time since the first until the last transaction (Normal transactions)	855	8	The shortest interval between two transactions (All transactions)	789
9	The minimum amount ever sent (Normal transactions)	788	10	The average amount received for each transaction (All transactions)	786
11	The average time between incoming transactions (All transactions)	768	12	The longest interval between two transactions (All transactions)	733
13	Time since the first until the last transaction (All transactions)	725	14	The average time between incoming transactions (Normal transactions)	712
15	The average incoming amount per day (All transactions)	710	16	The proportion of amount received by normal transactions out of all amount transferred	668
17	The longest interval between two transactions (Normal transactions)	635	18	The average incoming amount per day (Normal transactions)	613
19	The total amount transferred by outcoming and incoming transactions (All transactions)	581	20	The total amount ever received (Normal transactions)	503
21	The total amount transferred by outcoming and incoming transactions (Normal transactions)	499	22	The minimum amount ever sent (All transactions)	476
23	The average amount sent for each transaction (Normal transactions)	445	24	The minimum amount ever received (ERC20 transactions)	429
25	The total amount ever received (All transactions)	417	26	The total number of transactions (ERC20 transactions)	352
27	The total number of transactions (Normal transactions)	344	28	The shortest interval between two transactions (ERC20 transactions)	327
29	The average time between incoming transactions (ERC20 transactions)	324	30	The average amount sent for each transaction (All transactions)	289
31	The proportion of normal outcoming transactions out of all normal transactions	280	32	The total number of incoming transactions (Normal transactions)	254
33	The average time between outcoming transactions (All transactions)	232	34	The total number of transactions (Internal transactions)	223
35	The total amount ever sent (Normal transactions)	217	36	The longest interval between two transactions (ERC20 transactions)	211
37	0x73ff (bytecode)	190	38	The total amount ever sent (All transactions)	190
39	The average number of incoming transactions per day (Normal transactions)	183	40	Time since the first until the last transaction (ERC20 transactions)	179
41	The proportion of normal outcoming transactions out of all transactions	175	42	SWAP1 DUP2 MSTORE PUSH1 (opcodes and operands)	173
43	The proportion of all outcoming transactions out of all transactions	167	44	The average amount received for each transaction (ERC20 transactions)	158
45	The average time between outcoming transactions (ERC20 transactions)	158	46	The proportion of normal incoming transactions out of all transactions	151
47	The average incoming amount per day (ERC20 transactions)	146	48	The proportion of ERC20 incoming transactions out of all transactions	137
49	The total number of transactions (All transactions)	115	50	The average number of incoming transactions per day (All transactions)	114
51	The average amount transferred per day (Normal transactions)	111	52	The proportion of amount sent by ERC20 transactions out of all amount transferred	100
53	The proportion of amount received by ERC20 transactions out of all amount transferred	94	54	0x2 (operands)	88
55	The proportion of unique incoming address out of all incoming transactions	85	56	The total amount ever received (ERC20 transactions)	84
57	The proportion of ERC20 incoming transactions out of all ERC20 transactions	78	58	The average number of outcoming transactions per day (Normal transactions)	78
59	The proportion of normal transactions out of all transactions	76	60	The average number of transactions per day (Normal transactions)	74
61	The total amount transferred by outcoming and incoming transactions (ERC20 transactions)	70	62	The average number of transactions per day (All transactions)	70
63	The average number of outcoming transactions per day (All transactions)	70	64	The average incoming amount per hour (All transactions)	58
65	The average amount transferred per day (All transactions)	54	66	The proportion of internal incoming transactions out of all transactions	52
67	The average amount received for each transaction (Internal transactions)	48	68	The minimum amount ever sent (ERC20 transactions)	48
69	The proportion of unique incoming address out of all incoming transactions (Normal transactions)	48	70	The proportion of normal incoming transactions out of all normal transactions	46
71	The average incoming amount per hour (Normal transactions)	40	72	The average number of outcoming transactions per day (ERC20 transactions)	39
73	The average outcoming amount per day (All transactions)	38	74	swap1 (opcode)	34
75	The proportion of ERC20 transactions out of all transactions	27	76	The total number of incoming transactions (All transactions)	24
77	The average amount transferred per day (ERC20 transactions)	22	78	The proportion of amount sent by normal transactions out of all sent amount	18
79	The minimum amount ever received (Internal transactions)	17	80	The proportion of amount received by ERC20 transactions out of all sent amount	17
81	The number of unique outcoming addresses (Normal transactions)	17	82	The proportion of ERC20 outcoming transactions out of all transactions per day	16
83	The average time between incoming transactions (Internal transactions)	14	84	The proportion of incoming transactions out of all transactions	14
85	The average outcoming amount per day (Normal transactions)	13	86	calldatasize (opcode)	11
87	gt (opcode)	9	88	0xa9059cbb (operands)	8
89	sload (opcode)	8			

Table 10. The Top Features for Multi-class Classification (LightGBM)

No.	Features	Frequency	No.	Features	Frequency
1	The minimum amount ever received (All transactions)	4,521	2	The minimum amount ever received (Normal transactions)	4,334
3	JUMPDEST CALLVALUE ISZERO PUSH2 (opcode and operands)	3,783	4	0x73ff (bytecode)	3,597
5	The average amount received for each transaction (Normal transactions)	3,585	6	The shortest interval between two transactions (Normal transactions)	2,637
7	The average time between incoming transactions (All transactions)	2,404	8	0x60 (operands)	2,356
9	The proportion of amount sent by normal transactions out of all amount transferred	2,297	10	The average time between incoming transactions (Normal transactions)	2,229
11	The average time between outcoming transactions (Normal transactions)	2,071	12	Time since the first until the last transaction (Normal transactions)	2,069
13	The shortest interval between two transactions (All transactions)	1,993	14	The total number of transactions (Normal transactions)	1,762
15	The longest interval between two transactions (All transactions)	1,744	16	Time since the first until the last transaction (All transactions)	1,716
17	The minimum amount ever sent (Normal transactions)	1,677	18	The average amount received for each transaction (All transactions)	1,668
19	The longest interval between two transactions (Normal transactions)	1,656	20	push1 (opcode)	1,648
21	The proportion of amount received by normal transactions out of all amount transferred	1,648	22	The average incoming amount per day (Normal transactions)	1,352
23	The total amount ever received (Normal transactions)	1,323	24	The total number of transactions (ERC20 transactions)	1,261
25	0x2 (operands)	1,200	26	The total number of transactions (Internal transactions)	1,199
27	dup9 (opcode)	1,179	28	0x3fff (bytecode)	1,178
29	The average incoming amount per day (All transactions)	1,132	30	The minimum amount ever sent (All transactions)	1,120
31	The average amount sent for each transaction (Normal transactions)	1,043	32	JUMPDEST CALLVALUE DUP1 ISZERO (opcodes and operands)	1,031
33	0x8fc (operands)	1,021	34	calldataload (opcode)	1,012
35	The total amount transferred by outcoming and incoming transactions (Normal transactions)	986	36	The total amount transferred by outcoming and incoming transactions (All transactions)	946
37	dup1 (opcode)	939	38	0xfff1 (bytecode)	918
39	0xa0 (operands)	878	40	RETURN JUMPDEST CALLVALUE ISZERO (opcodes and operands)	856
41	SWAP1 DUP2 MSTORE PUSH1 (opcodes and operands)	795	42	The average time between outcoming transactions (All transactions)	760
43	sload (opcode)	741	44	0x57fe (bytecode)	707
45	0x40 MLOAD DUP1 DUP3 (opcodes and operands)	697	46	The proportion of unique incoming address out of all incoming transactions (Normal transactions)	692
47	The minimum amount ever received (ERC20 transactions)	625	48	The total number of incoming transactions (Normal transactions)	617
49	The total amount ever received (All transactions)	511	50	calldatasize(opcode)	506
51	JUMP JUMPDEST PUSH1 0x0 (opcodes and operands)	419	52	The average time between incoming transactions (ERC20 transactions)	407
53	The shortest interval between two transactions (ERC20 transactions)	390	54	The average number of incoming transactions per day (Normal transactions)	382
55	The proportion of normal outcoming transactions out of all transactions	339	56	The average time between outcoming transactions (Internal transactions)	333
57	The proportion of unique incoming address out of all incoming transactions (All transactions)	332	58	The average incoming amount per hour (Normal transactions)	323
59	mul (opcode)	295	60	The proportion of internal transactions out of all transactions	239
61	The average number of incoming transactions per hour (Normal transactions)	200	62	0x5b61 (bytecode)	198
63	The proportion of normal outcoming transactions out of all normal transactions	193	64	0x627a7a723058 (operands)	189
65	The proportion of normal incoming transactions out of all transactions	188	66	The average number of incoming transactions per day (All transactions)	185
67	revert (opcode)	183	68	dup8 (opcode)	182
69	The average amount sent for each transaction (All transactions)	179	70	The total number of transactions (All transactions)	171
71	The longest interval between two transactions (ERC20 transactions)	136	72	The average number of outcoming transactions per day (Normal transactions)	133
73	The number of unique outcoming addresses (Normal transactions)	129	74	The average number of outcoming transactions per day (All transactions)	119
75	The proportion of internal outcoming transactions out of all transactions	114	76	CALLVALUE DUP1 ISZERO PUSH2 (opcodes and operands)	110
77	The average incoming amount per hour (All transactions)	79	78	The average amount transferred per day (Normal transactions)	70
79	The average number of incoming transactions per hour (All transactions)	65	80	The average number of transactions per day (Normal transactions)	65
81	caller (opcode)	64	82	The proportion of ERC20 outcoming transactions out of all transactions	60
83	The total amount ever sent (Normal transactions)	59	84	The average time between outcoming transactions (ERC20 transactions)	57
85	dup7 (opcode)	57	86	The proportion of internal incoming transactions out of all transactions	56
87	0xa9059cbb (operands)	55	88	gt (opcode)	55
89	jumpdest (opcode)	54			

Table 11. The Top Features for Binary Classification (LightGBM)

No.	Features	Frequency	No.	Features	Frequency
1	The minimum amount ever received (All transactions)	1,174	2	The minimum amount ever received (Normal transactions)	860
3	The total number of transactions (Normal transactions)	697	4	The average amount received for each transaction (Normal transactions)	694
5	The average time between outcoming transactions (Normal transactions)	658	6	Time since the first until the last transaction (Normal transactions)	606
7	The minimum amount ever sent (Normal transactions)	576	8	The proportion of amount sent by normal transactions out of all transactions	568
9	The average time between incoming transactions (Normal transactions)	567	10	The average time between incoming transactions (All transactions)	522
11	The longest interval between two transactions (All transactions)	438	12	Time since the first until the last transaction (All transactions)	427
13	The shortest interval between two transactions (Normal transactions)	401	14	SWAP1 DUP2 MSTORE PUSH1 (opcodes and operands)	395
15	0x73ff (bytecode)	395	16	The shortest interval between two transactions (All transactions)	392
17	The total number of incoming transactions (Normal transactions)	384	18	The total number of transactions (ERC20 transactions)	382
19	The total amount transferred by outcoming and incoming transactions (Normal transactions)	380	20	The average amount received for each transaction (All transactions)	365
21	The minimum amount ever received (ERC20 transactions)	359	22	The proportion of amount received by normal transactions out of all amount transferred	348
23	The longest interval between two transactions (Normal transactions)	327	24	The proportion of normal outcoming transactions out of all normal transactions	308
25	The total amount ever received (Normal transactions)	300	26	The average time between incoming transactions (ERC20 transactions)	300
27	0x2 (operands)	294	28	The total number of transactions (Internal transactions)	288
29	The average time between outcoming transactions (ERC20 transactions)	265	30	The minimum amount ever sent (All transactions)	255
31	swap1 (opcode)	251	32	The average time between outcoming transactions (All transactions)	231
33	The average incoming amount per day (All transactions)	227	34	gt (opcode)	221
35	The total amount transferred by outcoming and incoming transactions (All transactions)	218	36	The average incoming amount per day (Normal transactions)	209
37	The shortest interval between two transactions (ERC20 transactions)	205	38	The average amount sent for each transaction (Normal transactions)	201
39	The proportion of ERC20 incoming transactions out of all transactions	175	40	The proportion of normal outcoming transactions out of all transactions	162
41	The proportion of all outcoming transactions out of all transactions	158	42	The proportion of internal incoming transactions out of all transactions	157
43	The proportion of normal incoming transactions out of all transcations	147	44	sha3 (opcode)	142
45	The proportion of unique incoming address out of all incoming transactions (All transactions)	135	46	The proportion of normal incoming transactions out of all normal transactions	132
47	calldatasize (opcode)	131	48	SJUMPDEST CALLVALUE DUP1 ISZERO (opcodes and operands)	122
49	The total amount ever received (All transactions)	121	50	The average number of incoming transactions per day (Normal transactions)	116
51	The longest interval between two transactions (ERC20 transactions)	112	52	The average amount received for each transaction (Internal transactions)	107
53	The proportion of amount sent by ERC20 transactions out of all amount transferred	99	54	The proportion of ERC20 incoming transactions out of all ERC20 transactions	91
55	The average number of incoming transactions per hour (Normal transactions)	80	56	0xa9059cbb (operands)	78
57	The proportion of unique incoming address out of all incoming transactions (Normal transactions)	70	58	RETURN JUMPDEST CALLVALUE ISZERO (opcodes and operands)	65
59	The total amount transferred by outcoming and incoming transactions (ERC20 transactions)	60	60	The average number of incoming transactions per day (All transactions)	59
61	The average number of outcoming transactions per day (Normal transactions)	57	62	The average amount received for each transaction (ERC20 transactions)	54
63	The average amount transferred per day (Normal transactions)	54	64	0x5b61 (bytecode)	52
65	The average incoming amount per hour (Normal transactions)	50	66	The average incoming amount per hour (All transactions)	48
67	The proportion of ERC20 outcoming transactions out of all transactions	48	68	The average amount sent for each transaction (All transactions)	46
69	The average time between incoming transactions (Internal transactions)	43	70	The minimum amount ever received (Internal transactions)	42
71	The average number of transactions per day (Normal transactions)	41	72	call (opcode)	39
73	The total number of incoming transactions (All transactions)	36	74	The proportion of all incoming transactions out of all transactions	36
75	The total number of transactions (All transactions)	29	76	The minimum amount ever sent (ERC20 transactions)	28
77	The average number of outcoming transactions per day (All transactions)	27	78	The average incoming amount per day (ERC20 transactions)	25
79	Time since the first until the last transaction (ERC20 transactions)	24	80	0x100 (operands)	24
81	The proportion of normal transactions out of all transactions	24	82	The proportion of ERC20 transactions out of all transactions	24
83	The average amount transferred per hour (Normal transactions)	23	84	The average incoming amount per day (Internal transactions)	17
85	The average amount sent for each transaction (ERC20 transactions)	14	86	The average number of outcoming transactions per hour (All transactions)	14
87	The total number of transactions (ERC721 transactions)	13	88	0x8fc (operands)	13
89	0x3 (operands)	13	90	The average outcoming amount per day (All transactions)	13
91	The average number of incoming transactions per hour (All transactions)	13	92	The proportion of ERC20 outcoming transactions out of all ERC20 transactions	12
93	The proportion of ERC20 outcoming transactions out of all outcoming transactions	12	94	returndatasize (opcode)	12
95	The proportion of normal incoming transactions out of all transactions	12	96	The proportion of amount sent by ERC20 transactions out of all sent amount	12
97	The average outcoming amount per day (Normal transactions)	12	98	The average number of outcoming transactions per hour (ERC20 transactions)	12
99	lt (opcode)	11	100	The total number of incoming transactions (ERC20 transactions)	11
101	The average amount transferred per day (All transactions)	11	102	The average number of outcoming transactions per day (ERC20 transactions)	11

ACKNOWLEDGMENTS

This work was funded and supported by Ericsson - Global artificial Intelligence accelerator in Montreal and Mitacs accelerate grant (IT16745).

REFERENCES

[1] The Root Cause Of Poly Network Being Hacked. 2022. Retrieved from https://slowmist.medium.com/the-root-cause-of-poly-network-being-hacked-ec2ee1b0c68f

[2] Choose a platform and step into the realm of blockchain. 2023. Retrieved from https://dataconomy.com/2022/06/best-blockchain-platforms-2022/
[3] Ethereum Whitepaper. 2022. Retrieved from https://ethereum.org/en/whitepaper/
[4] L. M. Goodman. 2014. Tezos— A self-amending crypto-ledger white paper. 4 (2014), 1432–1465. Retrieved from https://www.tezos.com/static/papers/whitepaper.pdf
[5] EOSIO Blockcian. 2022. Retrieved from https://eos.io/
[6] Cardano. 2022. Retrieved from https://cardano.org/
[7] David Voell, Frank Lu-Nick Gaski, Ram Jagadeesan, Renat Khasanshyn, Hart Montgomery, Stefan Teis, Tamas Blummer, Murali Krishna Katipalli, and Mic Bowman. 2016. Hyperledger whitepaper. Retrieved from https://wiki.hyperledger.org/groups/whitepaper/whitepaper-wg (2016).
[8] Lisk. 2022. Retrieved from https://lisk.com/
[9] Lexi Brent, Anton Jurisevic, Michael Kong, Eric Liu, Francois Gauthier, Vincent Gramoli, Ralph Holz, and Bernhard Scholz. 2018. Vandal: A scalable security analysis framework for smart contracts. *arXiv preprint arXiv:1809.03981* (2018).
[10] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: A static analysis framework for smart contracts. In *Proceedings of the IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB'19)*. IEEE, 8–15.
[11] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 254–269.
[12] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. 2018. Securify: Practical security analysis of smart contracts. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 67–82.
[13] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. Zeus: Analyzing safety of smart contracts. In *Network and Distributed System Security Symposium (NDSS'18)*. 1–12.
[14] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. SmartCheck: Static analysis of Ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. 9–16.
[15] Sunbeom So, Myungho Lee, Jisu Park, Heejo Lee, and Hakjoo Oh. 2020. VeriSmart: A highly precise safety verifier for Ethereum smart contracts. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'20)*. IEEE, 1678–1694.
[16] Yuchiro Chinen, Naoto Yanai, Jason Paul Cruz, and Shingo Okamura. 2020. RA: Hunting for re-entrancy attacks in Ethereum smart contracts via static analysis. In *Proceedings of the IEEE International Conference on Blockchain (Blockchain'20)*. IEEE, 327–336.
[17] Ting Chen, Rong Cao, Ting Li, Xiapu Luo, Guofei Gu, Yufei Zhang, Zhou Liao, Hang Zhu, Gang Chen, Zheyuan He, Yuxing Tang, Xiaodong Lin, and Xiaosong Zhang. 2020. SODA: A generic online detection framework for smart contracts. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'20)*.
[18] Jianbo Gao, Han Liu, Chao Liu, Qingshan Li, Zhi Guan, and Zhong Chen. 2019. Easyflow: Keep Ethereum away from overflow. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering (ICSE'19)*. IEEE, 23–26.
[19] Fuchen Ma, Ying Fu, Meng Ren, Mingzhe Wang, Yu Jiang, Kaixiang Zhang, Huizhong Li, and Xiang Shi. 2019. EVM: From offline detection to online reinforcement for Ethereum virtual machine. In *Proceedings of the IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER'19)*. IEEE, 554–558.
[20] Ivica Nikolić, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. 2018. Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 653–663.
[21] Shelly Grossman, Ittai Abraham, Guy Golan-Gueta, Yan Michalevsky, Noam Rinetzky, Mooly Sagiv, and Yoni Zohar. 2017. Online detection of effectively callback free objects with applications to smart contracts. *Proc. ACM Program. Lang.* 2, POPL (2017), 1–28.
[22] Michael Rodler, Wenting Li, Ghassan O Karame, and Lucas Davi. 2018. Sereum: Protecting existing smart contracts against re-entrancy attacks. *arXiv preprint arXiv:1812.05934* (2018).
[23] Chao Liu, Han Liu, Zhao Cao, Zhong Chen, Bangdao Chen, and Bill Roscoe. 2018. ReGuard: Finding reentrancy bugs in smart contracts. In *Proceedings of the 40th International Conference on Software Engineering*. 65–68.
[24] Steven Farrugia, Joshua Ellul, and George Azzopardi. 2020. Detection of illicit accounts over the Ethereum blockchain. *Expert Syst. Applic.* 150 (2020), 113318.
[25] Teng Hu, Xiaolei Liu, Ting Chen, Xiaosong Zhang, Xiaoming Huang, Weina Niu, Jiazhong Lu, Kun Zhou, and Yuan Liu. 2021. Transaction-based classification and detection approach for Ethereum smart contract. *Inf. Process. Manag.* 58, 2 (2021), 102462.

[26] Chaochen Shi, Yong Xiang, Jiangshan Yu, Longxiang Gao, Keshav Sood, and Robin Ram Mohan Doss. 2022. A bytecode-based approach for smart contract classification. In *Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER'22)*. IEEE, 1046–1054.

[27] Kazuki Hara, Takeshi Takahashi, Motoya Ishimaki, and Kazumasa Omote. 2021. Machine-learning approach using solidity bytecode for smart-contract honeypot detection in the Ethereum. In *Proceedings of the IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C'21)*. IEEE, 652–659.

[28] Liang Chen, Jiaying Peng, Yang Liu, Jintang Li, Fenfang Xie, and Zibin Zheng. 2020. Phishing scams detection in Ethereum transaction network. *ACM Trans. Internet Technol.* 21, 1 (2020), 1–16.

[29] Jiajing Wu, Qi Yuan, Dan Lin, Wei You, Weili Chen, Chuan Chen, and Zibin Zheng. 2020. Who are the phishers? Phishing scam detection on Ethereum via network embedding. *IEEE Trans. Syst., Man Cybern.: Syst.* 52, 2 (2020), 1156–1166.

[30] Shucheng Li, Fengyuan Xu, Runchuan Wang, and Sheng Zhong. 2021. Self-supervised incremental deep graph learning for Ethereum phishing scam detection. *arXiv preprint arXiv:2106.10176* (2021).

[31] Haixian Wen, Junyuan Fang, Jiajing Wu, and Zibin Zheng. 2021. Transaction-based hidden strategies against general phishing detection framework on Ethereum. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'21)*. IEEE, 1–5.

[32] Yijun Xia, Jieli Liu, and Jiajing Wu. 2022. Phishing detection on Ethereum via attributed ego-graph embedding. *IEEE Trans. Circ. Syst. II: Express Briefs* 69, 5 (2022), 2538–2542.

[33] Weili Chen, Zibin Zheng, Jiahui Cui, Edith Ngai, Peilin Zheng, and Yuren Zhou. 2018. Detecting Ponzi schemes on Ethereum: Towards healthier blockchain technology. In *Proceedings of the World Wide Web Conference*. 1409–1418.

[34] Yincheng Lou, Yanmei Zhang, and Shiping Chen. 2020. Ponzi contracts detection based on improved convolutional neural network. In *2020 IEEE International Conference on Services Computing (SCC)*. IEEE, 353–360.

[35] Yanmei Zhang, Siqian Kang, Wei Dai, Shiping Chen, and Jianming Zhu. 2021. Code will speak: Early detection of Ponzi smart contracts on Ethereum. In *Proceedings of the IEEE International Conference on Services Computing (SCC'21)*. IEEE, 301–308.

[36] Ali Aljofey, Qingshan Jiang, and Qiang Qu. 2021. A supervised learning model for detecting Ponzi contracts in Ethereum blockchain. In *Proceedings of the International Conference on Big Data and Security*. Springer, 657–672.

[37] Yanmei Zhang, Wenqiang Yu, Ziyu Li, Salman Raza, and Huaihu Cao. 2021. Detecting Ethereum Ponzi schemes based on improved LightGBM algorithm. *IEEE Trans. Computat. Soc. Syst.* 9, 2 (2021), 624–637.

[38] Ting Chen, Zihao Li, Yufei Zhang, Xiapu Luo, Ang Chen, Kun Yang, Bin Hu, Tong Zhu, Shifang Deng, Teng Hu, et al. 2019. DataEther: Data exploration framework for Ethereum. In *Proceedings of the IEEE 39th International Conference on Distributed Computing Systems (ICDCS'19)*. IEEE, 1369–1380.

[39] Xinming Wang, Jiahao He, Zhijian Xie, Gansen Zhao, and Shing-Chi Cheung. 2019. ContractGuard: Defend Ethereum smart contracts with embedded intrusion detection. *IEEE Trans. Serv. Comput.* 13, 2 (2019), 314–328.

[40] Mengya Zhang, Xiaokuan Zhang, Yinqian Zhang, and Zhiqiang Lin. 2020. TXSPECTOR: Uncovering attacks in Ethereum from transactions. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security'20)*. 2775–2792.

[41] Shlomi Linoy, Suprio Ray, and Natalia Stakhanova. 2021. EtherProv: Provenance-aware detection, analysis, and mitigation of Ethereum smart contract security issues. In *Proceedings of the IEEE International Conference on Blockchain (Blockchain'21)*. IEEE, 1–10.

[42] Christof Ferreira Torres, Antonio Ken Iannillo, Arthur Gervais, and Radu State. 2021. The eye of Horus: Spotting and analyzing attacks on Ethereum smart contracts. In *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 33–52.

[43] Lei Wu, Siwei Wu, Yajin Zhou, Runhuai Li, Zhi Wang, Xiapu Luo, Cong Wang, and Kui Ren. 2020. EthScope: A transaction-centric security analytics framework to detect malicious smart contracts on Ethereum. *arXiv preprint arXiv:2005.08278* (2020).

[44] Ramiro Camino, Christof Ferreira Torres, Mathis Baden, and Radu State. 2020. A data science approach for detecting honeypots in Ethereum. In *Proceedings of the IEEE International Conference on Blockchain and Cryptocurrency (ICBC'20)*. IEEE, 1–9.

[45] Weimin Chen, Xinran Li, Yuting Sui, Ningyu He, Haoyu Wang, Lei Wu, and Xiapu Luo. 2021. SadPonzi: Detecting and characterizing Ponzi schemes in Ethereum smart contracts. *Proc. ACM Measur. Anal. Comput. Syst.* 5, 2 (2021), 1–30.

[46] Jinhuan Wang, Pengtao Chen, Shanqing Yu, and Qi Xuan. 2021. TSGN: Transaction subgraph networks for identifying Ethereum phishing accounts. In *Third International Conference on Blockchain and Trustworthy Systems (BlockSys'21)*. Springer, 187–200.

[47] Wenhan Hou, Bo Cui, and Ru Li. 2022. Detecting phishing scams on Ethereum using graph convolutional networks with conditional random field. In *Proceedings of the IEEE 24th International Conference on High Performance Computing & Communications; 8th International Conference on Data Science & Systems; 20th International Conference*

on Smart City; 8th International Conference on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys'22). IEEE, 1495–1500.

[48] Jintao Luo, Jiwei Qin, Ruijin Wang, and Lu Li. 2023. A phishing account detection model via network embedding for Ethereum. *IEEE Trans. Circ. Syst. II: Express Briefs* (2023).

[49] Shucheng Li, Fengyuan Xu, Runchuan Wang, and Sheng Zhong. 2023. Self-supervised incremental deep graph learning for Ethereum phishing scam detection. In *ACM Conference on Multimedia*.

[50] Ákos Hajdu and Dejan Jovanović. 2020. solc-verify: A modular verifier for solidity smart contracts. In *11th International Conference on Verified Software. Theories, Tools, and Experiments (VSTTE'19)*. Springer, 161–179.

[51] Mythril classic. 2023. Retrieved from https://github.com/ConsenSys/mythril-classic/

[52] Weisong Sun, Guangyao Xu, Zijiang Yang, and Zhenyu Chen. 2020. Early detection of smart Ponzi scheme contracts based on behavior forest similarity. In *Proceedings of the IEEE 20th International Conference on Software Quality, Reliability and Security (QRS'20)*. IEEE, 297–309.

[53] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. 2016. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928* (2016).

[54] Lu Liu, Lili Wei, Wuqi Zhang, Ming Wen, Yepang Liu, and Shing-Chi Cheung. 2021. Characterizing transaction-reverting statements in Ethereum smart contracts. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE'21)*. IEEE, 630–641.

[55] Hyochang Baek, Junhyoung Oh, Chang Yeon Kim, and Kyungho Lee. 2019. A model for detecting cryptocurrency transactions with discernible purpose. In *Proceedings of the 11th International Conference on Ubiquitous and Future Networks (ICUFN'19)*. IEEE, 713–717.

[56] Sijia Li, Gaopeng Gou, Chang Liu, Chengshang Hou, Zhenzhen Li, and Gang Xiong. 2022. TTAGN: Temporal transaction aggregation graph network for Ethereum phishing scams detection. In *Proceedings of the ACM Web Conference*. 661–669.

[57] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. 1984. Classification and regression trees. wadsworth int. *Group* 37, 15 (1984), 237–251.

[58] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Vol. 1. IEEE, 278–282.

[59] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan Xie, Min Lin, Yifeng Geng, and Yutian Li. 2015. XGBoost: Extreme gradient boosting. *R Package Version 0.4-2* 1, 4 (2015), 1–4.

[60] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. GBM: A highly efficient gradient boosting decision tree. *Adv. Neural Inf. Process. Syst.* 30 (2017).

[61] Jian-Wei Liao, Tsung-Ta Tsai, Chia-Kang He, and Chin-Wei Tien. 2019. SoliAudit: Smart contract vulnerability assessment based on machine learning and fuzz testing. In *Proceedings of the 6th International Conference on Internet of Things: Systems, Management and Security (IOTSMS'19)*. IEEE, 458–465.

[62] Massimo Bartoletti, Salvatore Carta, Tiziana Cimoli, and Roberto Saia. 2020. Dissecting Ponzi schemes on Ethereum: Identification, analysis, and impact. *Fut. Gen. Comput. Syst.* 102 (2020), 259–277.

[63] Xuezhe He, Tan Yang, and Liping Chen. 2022. CTRF: Ethereum-based Ponzi contract identification. *Secur. Commun. Netw.* 2022 (2022).

[64] honeybadger. 2022. Retrieved from https://honeybadger.uni.lu/

[65] Christof Ferreira Torres, Mathis Steichen, and Radu State. 2019. The art of the scam: Demystifying honeypots in Ethereum smart contracts. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security'19)*. 1591–1607.

[66] Nitesh Kumar, Ajay Singh, Anand Handa, and Sandeep Kumar Shukla. 2020. Detecting malicious accounts on the Ethereum blockchain with supervised learning. In *Proceedings of the 4th International Symposium on Cyber Security Cryptography and Machine Learning (CSCML'20)*. Springer, 94–109.

[67] Etherscan APIs documentation. 2023. Retrieved from https://docs.etherscan.io/

[68] pyevmasm–Ethereum Virtual Machine (EVM) disassembler and assembler. 2023. Retrieved from https://github.com/crytic/pyevmasm

[69] Weili Chen, Zibin Zheng, Edith C.-H. Ngai, Peilin Zheng, and Yuren Zhou. 2019. Exploiting blockchain data to detect smart Ponzi schemes on Ethereum. *IEEE Access* 7 (2019), 37575–37586.

[70] XGBoost Documentation. 2023. Retrieved from https://xgboost.readthedocs.io/

[71] LightGBM Documentation. 2023. Retrieved from https://lightgbm.readthedocs.io/

[72] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. *Adv. Neural Inf. Process. Syst.* 31 (2018).

[73] Tu Dinh Nguyen Dai Quoc Nguyen and Dinh Phung. 2019. Unsupervised universal self-attention network for graph classification. *arXiv preprint arXiv:1909.11855* (2019).
[74] Geng Li, Murat Semerci, Bulent Yener, and Mohammed J. Zaki. 2011. Graph classification via topological and label attributes. In *Proceedings of the 9th International Workshop on Mining and Learning with Graphs (MLG'11).*

Received 25 March 2023; revised 12 September 2023; accepted 18 November 2023