Hunting in the Dark Forest: A Pre-trained Model for On-chain Attack Transaction Detection in Web3

Zhiying Wu
Sun Yat-sen University
Guangzhou, China
wuzhy95@mail2.sysu.edu.cn

Jiajing Wu*
Sun Yat-sen University
GuangDong Engineering Technology Research Center of Blockchain
Zhuhai, China
wujiajing@mail.sysu.edu.cn

Hui Zhang
Sun Yat-sen University
Guangzhou, China
zhangh827@mail2.sysu.edu.cn

Zibin Zheng
Sun Yat-sen University
GuangDong Engineering Technology Research Center of Blockchain
Zhuhai, China
zhzibin@mail.sysu.edu.cn

Weiqiang Wang
Independent
Hangzhou, China
wang.weiqiang@gmail.com

Abstract

In recent years, a large number of on-chain attacks have emerged in the blockchain empowered Web3 ecosystem. In the year of 2023 alone, on-chain attacks have caused losses of over $585 million. Attackers use blockchain transactions to carry out on-chain attacks, for example, exploiting vulnerabilities or business logic flaws in Web3 applications. A wealth of efforts have been devoted to detecting on-chain attack transactions through expert patterns and machine learning techniques. However, in this ever-evolving ecosystem, the performance of current methods is limited in detecting new on-chain attacks, due to the obsoleting of attack recognition patterns or the reliance on on-chain attack samples. In this paper, we propose a universal approach for detecting on-chain attacks even when there are few or even no new on-chain attack samples. Specifically, an in-depth analysis of the transaction characteristics is conducted, and we propose a new insight to train a generic attack transaction detecting model, i.e., transaction reconstruction. Particularly, to overcome the over-fitting in the transaction reconstruction task, we use the web-scale function comments related to transactions as supervision information, rather than expert-confirmed labels. Experimental results demonstrate that the proposed approach surpasses the supervised state-of-the-art by 13% in AUC, with just 30 known on-chain attack samples. Moreover, without any known attack samples, our method can still detect new on-chain attacks in the wild (with a precision of 61.83%). Among attacks detected in the wild, we confirm 1,692 address poisoning attacks, a new type of on-chain attack targeting token holders. Our code is available at: https://github.com/wuzhy1ng/attack_trans_detection_www25.

CCS Concepts

• **Security and privacy** → **Web application security**; • **Applied computing** → *Digital cash.*

Keywords

Attack detection, Blockchain transaction analysis, Web3

ACM Reference Format:
Zhiying Wu, Jiajing Wu, Hui Zhang, Zibin Zheng, and Weiqiang Wang. 2025. Hunting in the Dark Forest: A Pre-trained Model for On-chain Attack Transaction Detection in Web3. In *Proceedings of the ACM Web Conference 2025 (WWW '25), April 28-May 2, 2025, Sydney, NSW, Australia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3696410.3714928

1 Introduction

Web3, conceived a distributed Internet of value without a reliable third party, has received extensive attention from industry and academia [12]. Massive Web3 applications are built atop blockchain trading systems [2, 36, 42]. Users can activate Web3 application business logic by initiating blockchain transactions. The business logic of Web3 applications is usually implemented through smart contracts, which are Turing-complete programs on the blockchain [42]. Consequently, the transaction information carried by blockchain transactions goes far beyond "money transfer" in transactions in traditional financial systems.

Transactions can be utilized to carry out on-chain attacks, resulting in significant financial losses, and these attacks can be quite covert. For example, by carrying out blockchain transactions, attackers exploit the smart contract vulnerabilities or Web3 business logic flaws to profit. According to a report [4], the monetary detriment resulting from on-chain attacks surpassed $585 million in just 2023. In traditional financial systems (e.g., banks), users engaging in transaction activities are required to furnish authentic identity information, aka the Know-Your-Customer (KYC) process. Utilizing authentic identity information, financial industry experts are capable of deducing the intent of transactions. However, due to

*Corresponding Author.

WWW '25, Sydney, NSW, Australia

ACM ISBN 979-8-4007-1274-6/25/04
https://doi.org/10.1145/3696410.3714928

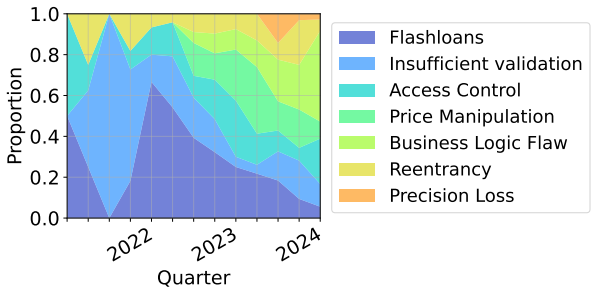


Figure 1: The changes in the proportions among several on-chain attack types over time.

the pseudonymity of blockchain accounts, users are not required to disclose their identities in order to engage in transactions. As a consequence, traditional technologies encounter challenges in scaling up to Web3 and blockchain ecosystems.

To ensure prompt reactions to the risks presented by on-chain attacks, some techniques [1, 19, 28, 38, 43, 48, 49, 51] have been proposed to autonomously detect attack transactions. However, new on-chain attacks continue to emerge, and existing methods are limited in identifying new types of on-chain attacks. We collect the on-chain attack reports from DeFiHackLab [35], and plot Figure 1, which illustrates the emergence of new attack types over time, e.g., price manipulation and precision loss. Moreover, in 2023, 23 of the 73 on-chain attacks did not fall in common established on-chain attack taxonomies [6, 26, 50], and these attacks resulted in losses, accounting for 52.5% of the total for the year [4]. According to another recent survey, only about 50% of new types of attacks [50] can be detected by current tools. On the one hand, existing techniques that rely on particular expert patterns may experience a gradual decline in effectiveness, as the attack vector continues to evolve. Especially when a new attack pattern appears, existing patterns may not be able to cover new attacks. On the other hand, while current learning-based approaches can extract attack knowledge automatically, their efficacy is contingent upon well-defined optimization goals, which usually depend on a substantial volume of labelled data or a clear attack taxonomy. Yet labeled on-chain attack samples and clear attack taxonomy are frequently absent for new attacks, which hinders the efficacy of learning-based methods.

In the Web3 ecosystem with diverse and rapidly evolving on-chain attacks, it is an imperative but challenging task to design a universal on-chain attack transaction detection model. ***First***, it is difficult to completely enumerate the patterns of on-chain attack transactions. Blockchain transactions contain heterogeneous data, e.g., traces, logs, and receipts, and attack patterns are reflected in the combination of these heterogeneous data [33, 38, 43]. Given that transaction data encompass a variety of attributes and categories, their combinations correspond to a space that is impossible to traverse within a limited time. ***Second***, labeled on-chain attack samples and a clear attack taxonomy are frequently absent for new on-chain attacks. While some work establishes taxonomies and labels samples for new attacks [47, 50], these efforts are frequently implemented subsequent to the occurrence of attacks. Prior to the occurrence of new attacks, it is exceedingly challenging for condensing attack features that have not yet materialized. And thus learning-based methods also struggle to learn generalizable decision boundaries for on-chain attack transaction detection directly.

This paper aims to address the aforementioned issues by proposing a generic, learning-based method for detecting on-chain attack transactions. First, we collect on-chain attack transactions from 2023 and earlier. Our analysis and prior work reveal that, at any given time, attack transactions exhibit some features (e.g., token leakage [32, 45], price slippage [43, 46]) that significantly distinguish them from non-attack transactions. And these features rarely appear in non-attack transactions. Based on the observation, we offer a new insight for designing on-chain attack transaction detection models: implementing pre-training to learn the features of reconstructing non-attack transactions. The pre-trained model can identify attack transactions using reconstruction error, without the need to pre-defined specific attack patterns to be detected. Moreover, we find that the generalizability of the pre-trained model can be limited by the potential parameter over-fitting, resulting from the reconstruction task. To prevent the over-fitting, the comment corresponding to functions triggered during transaction execution is used as supervision. This allows the model to learn meaningful embeddings in the reconstruction task. In this way, a generalizable on-chain attack transaction detection model can be developed, without any labeled attack transactions.

We prototype the proposed method on several EVM-compatible blockchains, e.g., Ethereum [42], BNBChain [2], and Polygon [36]. The conducted experiments evaluate the effectiveness and efficiency of our method. Firstly, for the attacks with known samples, e.g., reentrancy [33, 48], the proposed approach outperforms state-of-the-art full supervised methods by 13% in AUC, even if only 30-shot known attack samples are utilized. Secondly, without any known attack samples for training, the proposed method detect 1,707 real attacks in the wild. Especially, the detection results of the wild experiments reveal a new type of attack, i.e., address poisoning [31].

In summary, the major contributions are as follows:

- **Problem insights**. Despite differences in attack patterns, attack transactions often contain features that rarely appear in non-attack transactions. The observation inspires a new class of solutions: learning to reconstruct non-attack transactions and using reconstruction error to detect attack transactions, without the need to pre-define attack patterns or collect attack samples.
- **A pre-trained model**. We develop a pre-trained model capable of identifying attack transactions in the absence of known samples. During the pre-training, function comments corresponding to transactions are used as supervision, enhancing the model generalizability in the attack detection.
- **Experimental verification**. The proposed method outperforms the supervised state-of-the-art by 13% in AUC, although there are only 30 known on-chain attack samples. And we confirm that our method is efficient enough to deploy and monitor the real-time transaction flow. Moreover, without any known attack samples, the proposed method detects 1,707 on-chain attacks in the wild. Among detected attacks, 1,692 are confirmed as address poisoning attacks, a new type of on-chain attack.

2 Background and Related Work

In this section, we introduce some necessary concepts and related work of on-chain attack transaction detection task.

2.1 Terminology of Blockchains

EVM-compatible blockchain. Ethereum, one of the representatives of blockchain, is a blockchain trading system that enables developers to deploy Web3 applications. Ethereum Virtual Machine (EVM) [42] is a software environment that executes Web3 applications on Ethereum. The vast ecosystem of Ethereum drives multiple blockchain platforms, e.g., BNBChain, Polygon, to design compatible technologies for EVM, allowing deployment of the same applications as Ethereum without significant code modifications.
Accounts. Accounts can participate in transactions. Specifically, there are two kinds of accounts [42], i.e., externally owned accounts (aka. EOAs) and smart contracts. Externally owned accounts are controlled by users, while smart contract are controlled by on-chain program, supporting the logic of Web3 applications, e.g., issuing tokens. All accounts have a unique address as an identifier.
Transaction. Transactions are initiated by externally owned accounts. Every few seconds to minutes, new transactions are packaged into a block by the blockchain system and made publicly accessible. Transactions can be use to transfer native token (e.g., Ether in Ethereum) of a blockchain system from one account to another. Additionally, transactions can carry code or data that triggers the execution of on-chain programs.

2.2 On-chain Attack Detection

In order to carry out attack detection, existing methods focus on designing specific expert patterns and deep learning models.

2.2.1 Pattern-based Detection. In recent years, some researchers aim to design specific patterns to detect on-chain attack transactions [37, 39, 43, 46, 48, 51]. TXSPECTOR [48] constructs triggered the operation code into a graph, and uses the domain specific language to detect reentrancy, unchecked call, and suicidal attacks in transacitons. Zhou et al. [51] modeled the transaction execution as a graph, and defined six types of graph patterns to discover attack transactions, e.g., reentrancy, honeypot, etc. Wang et al. [39] summarized a flashloan pattern for each provider to identify flashloan transactions. Torres et al. [37] measured the three different types of frontrunning attacks: displacement, insertion, and suppression, based on their proposed heuristics. However, due to their design for specific tasks, pattern-based methods may lack generalizability and thus be difficult to scale to newly emerging attacks.

2.2.2 Learning-based Detection. Some learning-based methods [1, 19, 24, 33, 38, 44, 47] also attempted to extract transaction features and discover attack behaviors in transactions. DeFiScanner [38] designed a neural network to fuse features from external transactions and emitted events, which was then used to detect on-chain attacks. DEFIER [33] modeled the transaction execution process as a graph, and used graph embedding models and sequence embedding models to classify multi-stage attack transactions. Recent work [1, 19] has been devoted to extracting MEV (maximal extractable value) actions in transactions through deep learning methods. However, learning-based methods usually require a well-labelled dataset or the clear attack taxonomy for model training. Thus, it is difficult for existing learning-based methods to handle the new attack.

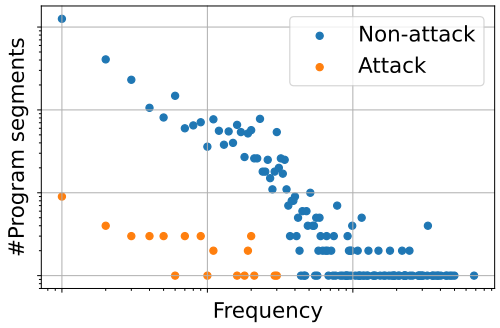


Figure 2: The frequency distribution of program segments.

3 Problem Analysis

This section conducts an analysis of the on-chain attack transaction task. Specifically, we explore the characteristics of attack and non-attack transactions, and provide new insights for designing a genic detection model.

3.1 Transaction Characteristics

In practice, existing technologies often model heterogeneous transaction data as transaction graphs [43, 44, 48, 51], allowing for modeling the associations between different data. Transaction graphs contain many high-order structures, which are subgraphs composed of multiple nodes that correspond to accounts and triggered program segments. These high-order structures can reflect the intent of transactions [43, 44, 51].

Some studies suggest that in non-attack transactions, the high-order structures reflecting complex intent are composed of a small number of high-order structure combinations [44]. One reason for this phenomenon may be the code reuse [34], where developers tend to use packaged code for constructing new programs. Additionally, due to the large volume of non-attack transactions, learning-based models can effectively learn the features of non-attack transactions [38, 44].

Human experts are able to identify and report new attacks. In fact, the high-order structures (e.g., token leakage [32, 45], price slippage [43, 46]), typically found in attacks, rarely appear in non-attack transactions. We conduct an empirical analysis using the DAppFL dataset [45], which collects both attack and non-attack transactions from 2023 and earlier. This dataset also indicates all triggered program segments in attack transactions and non-attack transactions. These program segments can be represented as subgraphs (also high-order structures), and the frequency distribution of different subgraphs is shown in Figure 2. It is evident that the high-order structures associated with attacks are less frequent and have a smaller number, displaying the traits of anomalous data.

> **Finding I**: In the transaction graph, high-order structures reflect transaction intent, and attack-related high-order structures are rare and infrequent.

3.2 Insights of Model Design

Drawing inspiration from Auto-Encoder [23] and the aforementioned observations, we can create a reconstruction task to train a model for detecting attack transactions. To be more precise, a vast number of non-attack transactions are used to teach the model how to encode and decode transaction features. When the model comes across an attack transaction, there can be a large reconstruction error, because the attack transaction includes rare high-order structures that contradict the data the model has learned. In this way, we can use the reconstruction error to filter out non-attack transactions and then detect attack transactions.

Note that just designing a reconstruction task is not enough. Prior work [52] suggests that the model may be over-fitting and fail to encode meaningful embeddings in the reconstruction task. This issue can result in the trained model lacking generalizability, making it difficult to distinguish between attack and non-attack transactions. To address this potential problem, we can use the comments corresponding to transaction-triggered functions as additional supervision, guiding the model to differentiate between different transactions, and thereby learn higher-quality embeddings. In fact, to facilitate users in checking the on-chain program security, some developers open source code and provide corresponding comments. In 98% of cases, these comments can accurately describe the true intent of a program [14]. However, not all on-chain programs involved in transactions have comments. More specifically, we random sampled over 10 million blocks from Ethereum, and only 14.54% of transactions trigger on-chain programs with comments. This is why we chose to use comments as the supervisory information rather than directly extracting transaction features from comments.

> **Finding II**: Reconstruction error shows promise in detecting attack transactions; however, to prevent over-fitting during training, extra supervision, e.g., comments, is required.

4 Proposed Approach

Based on aforementioned findings, we discuss the proposed approach in detail. Firstly, we conduct a transaction reconstruction task to pre-train the on-chain attack transaction detection model. The pre-trained model is also supervised by the comments for improved generalizability. Then, we demonstrate how to detect attack transactions using the pre-trained model.

4.1 Model Pre-training

Figure 3 illustrates the overall framework of the pre-training. The transaction data are modeled as heterogeneous graphs, whose features are extracted using a graph neural network (GNN) [30]. And we leverage a parameter-freezed language model to extract comment embeddings. Furthermore, we design two training tasks, i.e., **transaction reconstruction (TXR)** and **transaction-comment contrast (TCC)**, to jointly train the model parameters. Specifically, let $\mathcal{L}$ denotes the loss of the pre-training,

$$\mathcal{L} = \mathcal{L}_{TXR} + \mathcal{L}_{TCC}, \tag{1}$$

where $\mathcal{L}_{TXR}$ means the transaction reconstruction error, and $\mathcal{L}_{TCC}$ is the loss in the transaction-comment contrast.

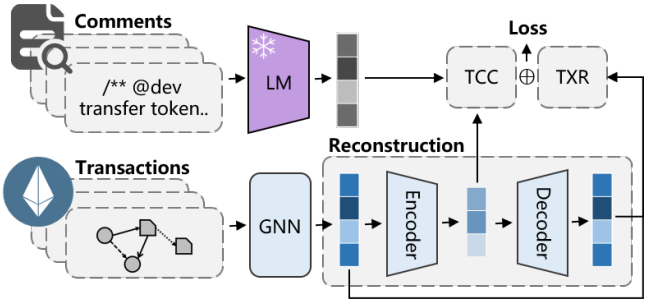


Figure 3: The pre-training framework.

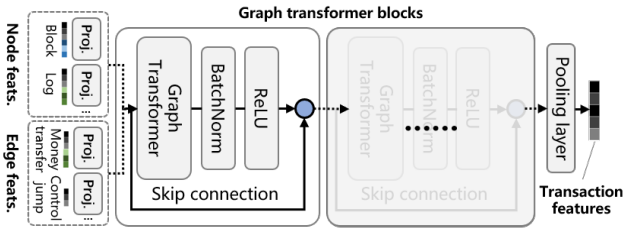


Figure 4: The framework of our GNN.

4.1.1 Transaction Reconstruction. Before conducting transaction reconstruction in the pre-training, we model the transaction data as graphs. Specifically, each transaction is modeled as a transaction graph, which is a heterogeneous graph. In the transaction graph, nodes represent accounts, contracts, basic blocks (i.e., executed program snippets), or logs, while edges represent various relationships between nodes, e.g., function calls, token transfers, log emissions, and more. Additionally, each type of node and edge has corresponding features. Appendix A.1 provides a detailed explanation of the transaction graph modeling.

Furthermore, we extract transaction features from the transaction graph. Prior studies [30] have demonstrated that the graph transformer can effectively extract information from graph data; therefore, we select it to extract features from transaction graphs. Note that [30] proposes a generic method for simple graphs. In this paper, the graph transformer is adapted to the scenarios discussed by considering the heterogeneous transaction data. Figure 4 illustrates the detail design of our GNN. Firstly, the different dimensions of nodes and edge features in the transaction graph make it difficult to combine different types of information. Therefore, we design projections that map all node features and edge features to the same dimension. Then, the graph transformer propagates node features and edge features, aiding in capturing high-order structure information hidden in the transaction graph. In the end, the pooling layer is used for reducing the whole graph as a vector, resulting in the feature representation for a given transaction:

(1) **Dimension projection**: Consider the node type set Γ_n, the edge type set Γ_e in the transaction graph, the type set is defined as:

$$\Gamma = \Gamma_n \cup \Gamma_e. \tag{2}$$

Let a type $t \in \Gamma$ has N_t corresponding elements and d_t demension features. Given the projection dimension $d \in \mathbb{N}_+$, the projection is:

$$\mathbf{H}_t = \mathbf{W}_t \mathbf{X}_t + \mathbf{b}_t, \tag{3}$$

where $\mathbf{H}_t \in \mathbb{R}^{d \times N_t}$ denotes the projected features, $\mathbf{W}_t \in \mathbb{R}^{d \times d_t}$ denotes a learnable type-specific transformation matrix for the type t, $\mathbf{X}_t \in \mathbb{R}^{d_t \times N_t}$ denotes the features of elements with type t, and $\mathbf{b}_t \in \mathbb{R}^{d \times 1}$ denotes the bias for the type t.

(2) **Feature propagation**: We design $l \in \mathbb{N}_+$ graph transformer blocks for feature propagation. Let $G(\cdot; \theta_g^{(i)})$ denotes i-th graph transformer layer parameterized by $\theta_g^{(i)}$. The output of the graph transformer in the i-th graph transformer block is:

$$\hat{\mathbf{H}}_n^{(i+1)} = G(\mathbf{H}_n^{(i)}, \mathbf{H}_e, \mathcal{E}; \theta_g^{(i)}), \tag{4}$$

where $\mathcal{E}$ is a set of edges in the transaction graph, $\mathbf{H}_n^{(i)}$ is the input node features in i-th graph transformer, and $\mathbf{H}_e = \{\bigcup_t \mathbf{H}_t | t \in \Gamma_e\}$ is edge features. Especially, for the 1-th graph transformer, the input node features are:

$$\mathbf{H}_n^{(1)} = \left\{ \bigcup_t \mathbf{H}_t | t \in \Gamma_n \right\}. \tag{5}$$

Additional, the batch normalization [17], ReLU activation [13], and skip connection [15] are conducted to avoid the over-smooth [18]. In this way, the output of the i-th graph transformer block is:

$$\mathbf{H}_n^{(i+1)} = Relu(BatchNorm(\hat{\mathbf{H}}_n^{(i+1)})) + \mathbf{H}_n^{(i)}. \tag{6}$$

(3) **Pooling**: We perform the average, mean, and max pooling operation on $\mathbf{H}_n^{(l+1)}$. Moreover, we concatenate all pooling output as the transaction features $\mathbf{z}_{tx}$, i.e.,

$$\mathbf{z}_{tx} = Avg(\mathbf{H}_n^{(l+1)}) \| Sum(\mathbf{H}_n^{(l+1)}) \| Max(\mathbf{H}_n^{(l+1)}), \tag{7}$$

where $\|$ means concatenation, Avg, Sum, and Max denote the operation for calculating the average, mean, and maximum of all dimensions in $\mathbf{H}_n^{(l+1)}$, respectively.

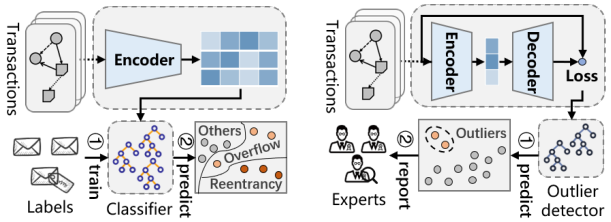
Transaction reconstruction encodes and then decodes transaction features, comparing differences between features before and after reconstruction, and minimizing reconstruction error to the greatest extent possible. In this paper, we use multi-layer perceptron as the encoder and decoder, following the practices of Auto-Encoder [23]. Let $E(\cdot; \theta_e)$ and $D(\cdot; \theta_d)$ are the encoder and decoder, respectively parameterized by θ_e and θ_d; the reconstructed transaction features $\hat{\mathbf{z}}_{tx}$ is:

$$\hat{\mathbf{z}}_{tx} = D(E(\mathbf{z}_{tx}; \theta_e); \theta_d). \tag{8}$$

In this way, the transaction reconstruction error is:

$$\mathcal{L}_{TXR} = ||\hat{\mathbf{z}}_{tx} - \mathbf{z}_{tx}||_2. \tag{9}$$

4.1.2 Transaction-Comment Contrast. This task contrasts the encoded transaction embeddings with comment features, ensuring that paired transactions and comments are close together, while unpaired ones are farther apart. Contrastive learning drives the model to generate meaningful embeddings under the supervision



(a) Detection with known samples (b) Detection without known samples

Figure 5: Use the pre-trained transaction embedding model, to perform on-chain attack detection.

of comments, rather than over-fitting to the training data. Consider an k-pair batch, the loss of the transaction-comment contrast is:

$$\mathcal{L}_{TCC} = (\mathcal{L}_{T2C} + \mathcal{L}_{C2T})/2, \tag{10}$$

where $\mathcal{L}_{T2C}$ denotes the contrastive loss from transactions to comments, and $\mathcal{L}_{C2T}$ denotes the contrastive loss from comments to transactions. Intuitively, matching the transaction-comment pair should be symmetrical: 1) For each transaction, it should be similar to the matched comment and dissimilar to other comments (i.e., $\mathcal{L}_{T2C}$); 2) For each comment, it should be similar to the matched transaction and dissimilar to other transactions (i.e., $\mathcal{L}_{C2T}$).

Let $T_i = E(\mathbf{z}_{tx}^i; \theta_e)$ denotes the embedding for the i-th transaction features $\mathbf{z}_{tx}^i$, and C_i means the embedding for the i-th comment. Note that we use a parameter-freezed language model, i.e., CodeBERT [10], to embed the comment, for it contains a wealth of pre-training knowledge that aids in understanding comments. The goal of $\mathcal{L}_{T2C}$ is to make the transaction embedding closer to the corresponding comment embedding, i.e.:

$$\mathcal{L}_{T2C} = -\frac{1}{k} \sum_{i=1}^{k} \log \left(\frac{\exp(sim(T_i, C_i)/\tau)}{\sum_{j=1}^{k} \exp(sim(T_i, C_j)/\tau)} \right), \tag{11}$$

where $sim(T_i, C_j)$ means the cosine similarity between i-th transaction embedding T_i and j-th comment embedding C_j. And $\mathcal{L}_{C2T}$ makes the document embedding closer to the corresponding transaction embedding, i.e.:

$$\mathcal{L}_{C2T} = -\frac{1}{k} \sum_{i=1}^{k} \log \left(\frac{\exp(sim(C_i, T_i)/\tau)}{\sum_{j=1}^{k} \exp(sim(C_i, T_j)/\tau)} \right), \tag{12}$$

in which $sim(C_i, T_j)$ denotes the cosine similarity between the comment embedding C_j and the transaction embedding T_i. A learnable temperature parameter $\tau \in \mathbb{R}$ is also provided in Equation 11 and Equation 12, to control the range of the logits.

Note that two or more transactions in a batch may involve the same comment, when the transaction-comment pair is randomly sampled. Consequently, employing the aforementioned loss functions could potentially impede the convergence of the optimization. Therefore, we incorporate constraints for each batch throughout the training procedure to guarantee the absence of duplicate comments.

4.2 On-chain Attack Transaction Detection

After the pre-training, the pre-trained model can be used to detect on-chain attack transactions. In practice, the detection can be

separated into two scenarios: detection when some of the attack samples are known and detection in the absence of attack samples.

4.2.1 Detection with Known Attack Samples. When it comes to some of the attack samples are known, the transaction embeddings can be utilized for supervised learning, leading to the development of an attack detection classifier. For instance, some well-studied attack patterns have been labeled with attack samples [38, 51]. Consequently, we can opt for classic classifiers, e.g., random forest [5], to conduct training on the known attack samples derived from well-studied attack patterns. Then the trained classifier can be applied to on-chain attack detection tasks. Figure 5(a) demonstrates how to detect on-chain attack transactions with known attack samples.

4.2.2 Detection without Known Attack Samples. In some cases, there are no available known samples for detecting attack transaction, especially when detecting the ones with newly emerging attack patterns. Nevertheless, based on aforementioned findings, we can still use the reconstruction error to detect attack transactions in the absence of known samples. Specifically, by leveraging the pre-trained model, every transaction can be assigned a reconstruction error. And a outlier detection method [21] can identify transactions with a large of reconstruction error. These identified transactions may be related to on-chain attacks. For further verification, experts are notified of these identified transactions, and assess whether on-chain attacks are involved.

5 Experiments

In this section, we evaluate the proposed method. Specifically, the experiments are designed to answer the following questions:

(**Q1**) Is the proposed method outperform existing approaches?

(**Q2**) Does the pre-training contribute to the performance improvement in on-chain attack detection?

(**Q3**) Can the proposed method be deployed in the wild and detect newly emerging attack transactions, even in the absence of known attack samples?

5.1 Experimental Settings

5.1.1 Data Collection. We randomly sample 1 million transactions from 18 million Ethereum blocks for the pre-training. Note that all sampled transaction trigger functions and the corresponding comments are publicly available on Etherscan. For each transaction, we use the comment corresponding to the first function it triggers as supervisory information. Since the first triggered function is specified by the user, it often provides higher-level semantic information [44]. In order to facilitate the comparison of the proposed method with existing approaches, we integrate the on-chain attack transactions dataset from current research [51], on-chain attack reports of DeFiHackLabs [35], and BlockSec [4]. Additionally, we use BlockchainSpider[1], an RPC-based transaction data collection toolkit, to collect the heterogeneous transaction data.

5.1.2 Compared Methods. We select state-of-the-art approaches, i.e., DeFiScanner [38], DEFIER [33], MoTS [44], and MetaSuites [3], for comparison experiments. Note that all these selected methods require a supervised training. Specifically, DeFiScanner fuses the global and local features of a transaction and then processes the fused features with LSTM [16] to train the model. DEFIER uses vertex embedding to work on the nodes in the transaction graph and graph embedding methods to work on the transaction graph itself. It then combines these two features and uses LSTM to work on them. MoTS calculates the network motifs contained in the transaction graph and uses the counting vector as the transaction representation. Additionally, a Web3 transaction analysis tool, MetaSuites, are taken into consideration. Based on ChatGPT, MetaSuites parses transaction intent using data from blockchain browsers, e.g., Etherscan[2], without requiring training samples. Note that the recent proposed BlockGPT [11] has demonstrated its ability of detecting on-chain attacks. Nevertheless, due to its under-review status and the absence of reproducible details, BlockGPT is not utilized as a comparative method in this paper.

5.1.3 Parameter Settings. In the experiment, referring to the prior work [30], we use 6 graph transformer blocks and 384 projected dimensions in GNN. In addition, the pre-training has 10 epoch, employs the 64-pair batch, and applies AdamW [22] for optimization. All experiments are repeated 10 times and report the average metrics, excluding the evaluation in the wild. Moreover, our experiments are conducted on a workstation with 512 GB of RAM, an Intel(R) Xeon(R) Gold 6148 CPU @ 2.40 GHz, and a GPU of NVIDIA GeForce RTX 3090 running on Ubuntu 20.04.1.

5.2 Comparative Experiment

In this section, we conduct experiments to answer **Q1**. Most existing methods are used to detect well-studied on-chain attacks with known samples. Therefore, in the comparative experiment, we selected five well-studied attack categories from the collected attack transactions, including reentrancy, honeypot, flashloan, integer overflow, and call injection. For the fairness of the comparative experiment, we follow the dataset construction strategy in prior work [38], that is, mix on-chain attack transactions into randomly selected non-attack transactions, guaranteeing that the proportion of attack transactions is lower than 5%. In the case of methods that necessitate supervised training, the dataset is arbitrarily partitioned into training and testing sets in a ratio of 8:2. In the case of methods that do not undergo supervised training, testing is conducted on the whole dataset. In particular, to demonstrate that the proposed method can also detect attack transactions in few known samples, we try to provide few-shot known attack transactions on each attack category for training a random forest (see Figure 5(a)) and perform testing on the remaining dataset.

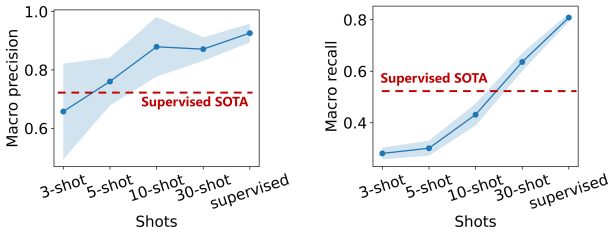
Table 1 shows the result of our comparative experiments. The proposed method outperforms all compared methods in terms of AUC metrics, both under full supervision and with only 30-shot known attack samples. Note that the precision and recall of all methods except our method in detecting honeypot attacks are 0. Honeypot [33] means using bait with no real paybacks to get someone to send some assets and then take part. And honeypot is typically implemented by on-chain program (i.e., smart contracts). Therefore, detecting honeypot attacks has to utilize program features to make a precise determination; the majority of the aforementioned

[1] https://github.com/wuzhy1ng/BlockchainSpider

[2] https://etherscan.io

Table 1: Results of on-chain attack detection. P and R denote precision and recall, respectively. K-shot means that there are K known samples for each kind of attack. For each group, the highest score is bolded, and the second highest score is underlined.

Supervised	Method	Reentrancy		Honeypot		Flashloan		Integer overflow		Call injection		AUC
		P	R	P	R	P	R	P	R	P	R	
Full	DeFiScanner	**1.00**	0.83	0.00	0.00	0.85	0.32	**1.00**	0.65	0.67	**1.00**	0.78
	DEFIER	0.00	0.00	0.00	0.00	**1.00**	0.03	0.36	0.45	**1.00**	0.05	0.54
	MoTS	0.99	0.70	0.00	0.00	0.50	**0.40**	0.83	0.11	0.00	0.00	0.64
	Ours	**1.00**	**0.95**	**0.74**	**0.35**	0.85	0.36	0.98	**0.70**	0.95	0.70	**0.94**
Few/Zero-shot	ChatGPT (MetaSuites)	0.00	0.00	0.00	0.00	**1.00**	0.15	0.00	0.00	0.00	0.00	0.515
	Ours (10-shot)	**1.00**	0.66	**0.79**	0.07	0.51	0.16	**0.99**	0.29	**0.99**	0.37	0.74
	Ours (30-shot)	**1.00**	**0.84**	0.73	**0.38**	0.39	**0.25**	**0.99**	**0.30**	0.95	**0.85**	**0.91**



(a) Macro precision and shots (b) Macro recall and shots

Figure 6: The variation of macro precision and macro recall with different shots in on-chain attack detection. The solid line represents the average metrics of repeated experiments.

methods do not. Moreover, because MetaSuites has not recognize honeypot, integer overflow, and call injection attacks, we mark the corresponding metrics as 0.

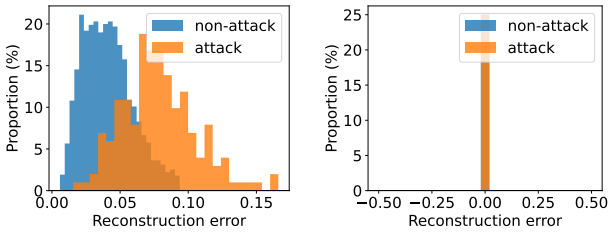
We further explore the impact of the known sample quantity on the performance of the proposed model. Figure 6(a) (resp. Figure 6(b)) illustrates the variations in macro precision (resp. macro recall) in on-chain attack detection tasks as the number of shots changes. For the proposed model, macro precision and macro recall significantly improve as the number of shots rises. Furthermore, we annotate the best results from the comparative methods with red dashed lines. Despite some fluctuations in the performance of the proposed method with few known on-chain attack samples, at 10-shot, both macro precision and macro recall are competitive the best results from the comparative methods.

5.3 Ablation Study

In this section, we conduct experiments to answer **Q2**. Our method design focuses on enhancing the efficacy of on-chain attack detection through the utilization of two pre-training tasks, i.e., transaction reconstruction and transaction-comment contrast. Therefore, we attempt to remove the key components proposed separately and observe the changes in metrics. Note that for the purposes of analysis, the dataset from the comparative experiment is still utilized in the ablation study. Then we use macro metrics to summarize the performance of the model in detecting on-chain attacks.

Table 2: Variations in the efficacy of the proposed method as TXR or TCC are eliminated. ΔF1-Score↓ indicates the decrease in macro F1 score.

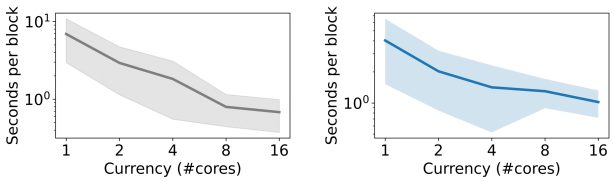
Method	Precision	Recall	F1-Score	ΔF1-Score↓
Ours	0.93±0.03	0.81±0.01	0.85±0.02	-
w/o TXR	0.91±0.03	0.70±0.03	0.77±0.03	-9.1%
w/o TCC	0.46±0.07	0.23±0.01	0.33±0.02	-61.4%



(a) Use TCC (b) Remove TCC

Figure 7: The distribution of the reconstruction error when use/remove ransaction-comment contrast (TCC).

Table 2 shows the result of ablation study. Whether removing transaction reconstruction or transaction-comment contrast, the performance of the proposed method decreases. Therefore, the pre-training does contribute to the efficacy of the proposed model in on-chain attack detection. Note that when without transaction-comment contrast, the macro F1 score of the proposed method significantly decreases. For more explanation, we calculate the distribution of reconstruction error for attack and non-attack transactions. Figure 7(a) shows that when the transaction-comment contrast is used, the reconstruction error for the majority of non-attack transactions is smaller than that for attack transactions. However, when the transaction-comment contrast is removed, there is almost no difference in reconstruction error between attack and non-attack transactions (Figure 7(b)). This also confirms the aforementioned insight that an over-fitting lies in pre-trained models, if there is no extra supervision in the transaction reconstruction task.



(a) Transaction graph modeling (b) Model inferring

Figure 8: Two time-consuming stages. The solid line represents the average time cost, while the area around the solid line represents the fluctuation range of time cost.

5.4 Detection in the Wild

Next, we conduct experiments to answer **Q3**. For the detection in the wild, the time required for detection should be shorter than the time taken for block generation. Otherwise, the deployed method can not be able to detect potential attacks in the latest transactions. Therefore, we evaluated the efficiency of the proposed method. As shown in Figure 8(a) and 8(b), when using 16 CPU cores, the two most time-consuming operations can be completed in a total of 1.62 seconds, which is shorter than the block generation time for two well-known blockchains, Ethereum (12s) and BNBChain (3s). Thus, the proposed method is efficient enough.

Then we perform a detection in the wild as depicted in Figure 5(b). The classical outlier detector, isolation forest, is used to identify the outliers with a large reconstruction error. Specifically, we conduct a continuous 31-day on-chain attack transaction detection on the Ethereum in January 2024. 2,761 suspected transactions are identified. And we have convened 3 non-author domain experts to evaluate suspicious transactions. Domain experts are allowed to use any method to evaluate suspicious transactions, including search engines, existing on-chain attack detection tools, blockchain browsers, etc. Three experts assess each transaction to determine if it is subject to an on-chain attack. The decision is made based on a majority vote. Note that when experts determine that a transaction is not associated with an attack, it is imperative to analyze the genuine purpose of the transaction and document the outcomes of the analysis. In this way, experts spend approximately 60 hours reviewing all suspicious transactions.

Table 3 demonstrates the result of manual analysis of on-chain attack detection in the wild. Among suspicious transactions, 1,707 transactions are confirmed as on-chain attacks, with an precision of 61.83%. Out of the confirmed on-chain attacks, 0.07% are exploiting contract vulnerabilities, 0.47% are rug-pulls, and 61.28% are instances of address poisoning. To be clarify, two detected attacks, involving exploiting contract vulnerability, have been confirmed by security organization [4, 35]. For example, there is one attack[3] targeting the bussiness logic flaw in Socket project[4], where the hacker profited approximately $2.5 million in an on-chain attack transaction. Furthermore, current research has demonstrated the emergence of rug-pull as a new form of on-chain attack in recent years, resulting in financial losses surpassing $2.8 billion [20]. The proposed method also detects a large number of undisclosed address poisoning attacks (61.28%), which is a new kind of attack towards blockchain users [31]. Further analysis on the experiment in the wild can be found in Appendix A.3. To summarize, the confirmed on-chain attacks demonstrate that the proposed method is capable of detecting new attacks, even when no known attack samples are available in the training.

[3]Developers of Socket confirmed this attack. See https://twitter.com/SocketDotTech/status/1747349422730813525.
[4]https://twitter.com/SocketDotTech

Table 3: Manual estimation of the on-chain attack detection result in the wild.

Category	Is attack?	# Transaction	(%)
Address poisoning	✓	1,692	61.28%
Rug pull	✓	13	0.47%
Exploiting	✓	2	0.07%
Batch transfer	✗	550	19.92%
Mining	✗	407	14.74%
Arbitrage	✗	15	0.54%
Airdrop	✗	6	0.22%
Others	✗	76	2.75%
Total	-	2,761	100%

6 Conclusion

In this work, we proposed a universal approach for detecting on-chain attack transactions in Web3. We first analyzed characteristics of transactions, and then introduced a general purpose pre-trained model to detect on-chain attack transactions. The proposed pre-trained model can be applied to newly emerging on-chain attacks and is independent of expert patterns and known attack samples. Specifically, the pre-training consists of two tasks, i.e., transaction reconstruction and transaction-comment contrast. In terms of the transaction reconstruction, we leveraged the graph neural network to extract transaction features and trained the model parameters in reconstructing transaction features. To prevent the over-fitting in the transaction reconstruction, we utilized the comment corresponding to functions triggered during transaction execution as supervision, i.e., transaction-comment contrast. Experiments demonstrated that the proposed model is effective and efficient. When there are only 30 known on-chain attack samples, the proposed method can still outperform current methods. And we found that the proposed method is fast enough to be deployed. Without known attack samples, the proposed method also detected attacks in the wild, with a precision of 61.83%. Especially, 1,692 of the expert-confirmed attacks in the wild are address poisoning attacks, which is a new type of attack. Note that this paper have not discuss the model hyperparameter selection, because it is orthogonal to our main contributions. Therefore, we will further delve into these aspects in future work. In summary, this paper explores a *new* approach to designing on-chain attack detection models, with the aim of providing new insights for enhancing Web3 security.

Acknowledgments

The work described in this paper is supported by the National Natural Science Foundation of China (62332004 and 62372485), the Natural Science Foundation of Guangdong Province (2023A1515011314), the Fundamental Research Funds for the Central Universities of China (24lgqb018). Moreover, we would like to thank Qingqing Sun[5], Jun Chen[6], Bowen Song[7] and Lun Zhang[8], whose expertise was invaluable in formulating the research questions and methodology. Their insightful feedback pushed us to sharpen our thinking and brought our work to a higher level.

References

[1] Kushal Babel, Mojan Javaheripi, Yan Ji, Mahimna Kelkar, Farinaz Koushanfar, and Ari Juels. 2023. Lanturn: Measuring economic security of smart contracts through adaptive learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, Copenhagen, Denmark, 1212–1226.

[2] Binance. 2020. *BNB Chain, Build Web3 dApps on the Most Popular Blockchain.* Retrieved October 12, 2023 from https://www.bnbchain.org/

[3] BlockSec. 2024. MetaSuites, The Swiss Army Knife for Builders. https://blocksec.com/metasuites.

[4] Blocksec. 2024. Phalcon: Security Incidents. https://phalcon.blocksec.com/explorer/security-incidents.

[5] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[6] Chainalysis. 2024. The Chainalysis 2024 Crypto Crime Report: Crypto Hacking Stolen Funds. https://www.chainalysis.com/blog/crypto-hacking-stolen-funds-2024.

[7] Ting Chen, Zihao Li, Xiapu Luo, Xiaofeng Wang, Ting Wang, Zheyuan He, Kezhao Fang, Yufei Zhang, Hang Zhu, Hongwei Li, et al. 2021. SigRec: Automatic recovery of function signatures in smart contracts. *IEEE Transactions on Software Engineering* 48, 8 (2021), 3066–3086.

[8] 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 2018 world wide web conference*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 1409–1418.

[9] Vogelsteller Fabian and Buterin Vitalik. 2015. ERC20: Token Standard. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md.

[10] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP*. Association for Computational Linguistics, Online, 1536–1547.

[11] Yu Gai, Liyi Zhou, Kaihua Qin, Dawn Song, and Arthur Gervais. 2023. Blockchain large language models. *arXiv preprint arXiv:2304.12749* (2023).

[12] Sam Gilbert. 2022. *Crypto, web3, and the Metaverse.* Retrieved July 8, 2022 from https://www.bennettinstitute.cam.ac.uk/wp-content/uploads/2022/03/Policy-brief-Crypto-web3-and-the-metaverse.pdf

[13] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR, Fort Lauderdale, FL, USA, 315–323.

[14] Sicheng Hao, Yuhong Nan, Zibin Zheng, and Xiaohui Liu. 2023. SmartCoCo: Checking Comment-Code Inconsistency in Smart Contracts via Constraint Propagation and Binding. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 294–306. doi:10.1109/ASE56229.2023.00142

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. IEEE, Las Vegas, NV, USA, 770–778.

[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[17] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. ACM, Lille, France, 448–456.

[18] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *Proceedings of the IEEE/CVF international conference on computer vision*. IEEE, Montreal, BC, Canada, 9267–9276.

[19] Zihao Li, Jianfeng Li, Zheyuan He, Xiapu Luo, Ting Wang, Xiaoze Ni, Wenwu Yang, Xi Chen, and Ting Chen. 2023. Demystifying DeFi MEV Activities in Flashbots Bundle. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, Copenhagen, Denmark, 165–179.

[20] Zewei Lin, Jiachi Chen, Zibin Zheng, Jiajing Wu, Weizhe Zhang, and Yongjuan Wang. 2024. CRPWarner: Warning the Risk of Contract-related Rug Pull in DeFi Smart Contracts. *arXiv preprint arXiv:2403.01425* (2024).

[21] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *IEEE international conference on data mining*. IEEE, Pisa, Italy, 413–422.

[22] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[23] Umberto Michelucci. 2022. An introduction to autoencoders. *arXiv preprint arXiv:2201.03898* (2022).

[24] Peng Qian, Jianting He, Lingling Lu, Siwei Wu, Zhipeng Lu, Lei Wu, Yajin Zhou, and Qinming He. 2023. Demystifying Random Number in Ethereum Smart Contract: Taxonomy, Vulnerability Identification, and Attack Detection. *IEEE Transactions on Software Engineering* 49, 7 (2023), 3793–3810.

[25] Kaihua Qin, Zhe Ye, Zhun Wang, Weilin Li, Liyi Zhou, Chao Zhang, Dawn Song, and Arthur Gervais. 2023. Towards automated security analysis of smart contracts based on execution property graph. *arXiv preprint arXiv:2305.14046* (2023).

[26] SWC Registry. 2020. Smart Contract Weakness Classification (SWC). https://swcregistry.io/.

[27] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[28] Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi. 2019. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. In *Annual Network and Distributed System Security Symposium*. The Internet Society, San Diego, California, USA.

[29] Michael Rodler, Wenting Li, Ghassan O Karame, and Lucas Davi. 2021. {EVMPatch}: Timely and automated patching of ethereum smart contracts. In *USENIX Security Symposium*. USENIX Association, Vancouver, B.C., Canada, 1289–1306.

[30] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. 2021. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. International Joint Conferences on Artificial Intelligence Organization, Montreal-themed virtual reality, 1548–1554.

[31] Guan Shixuan and Li Kai. 2024. Characterizing Ethereum Address Poisoning Attack. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*.

[32] Jianzhong Su, Xingwei Lin, Zhiyuan Fang, Zhirong Zhu, Jiachi Chen, Zibin Zheng, Wei Lv, and Jiashui Wang. 2023. DeFiWarder: Protecting DeFi Apps from Token Leaking Vulnerabilities. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1664–1675.

[33] Liya Su, Xinyue Shen, Xiangyu Du, Xiaojing Liao, XiaoFeng Wang, Luyi Xing, and Baoxu Liu. 2021. Evil under the sun: understanding and discovering attacks on Ethereum decentralized applications. In *USENIX Security Symposium*. USENIX Association, Vancouver, B.C., Canada, 1307–1324.

[34] Kairan Sun, Zhengzi Xu, Chengwei Liu, Kaixuan Li, and Yang Liu. 2023. Demystifying the Composition and Code Reuse in Solidity Smart Contracts. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (San Francisco, CA, USA). Association for Computing Machinery, New York, NY, USA, 796–807. doi:10.1145/3611643.3616270

[35] SunWeb3Sec. 2023. DeFi Hacks Reproduce - Foundry. https://github.com/SunWeb3Sec/DeFiHackLabs.

[36] Polygon Team. 2019. *Bring the World to Ethereum, Polygon.* Retrieved October 12, 2023 from https://polygon.technology/

[37] Christof Ferreira Torres, Ramiro Camino, et al. 2021. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *USENIX Security Symposium*. USENIX Association, Vancouver, B.C., Canada, 1343–1359.

[38] Bin Wang, Xiaohan Yuan, Li Duan, Hongliang Ma, Chunhua Su, and Wei Wang. 2024. DeFiScanner: Spotting DeFi Attacks Exploiting Logic Vulnerabilities on Blockchain. *IEEE Transactions on Computational Social Systems* 11, 2 (2024), 1577–1588.

[39] Dabao Wang, Siwei Wu, Ziling Lin, Lei Wu, Xingliang Yuan, Yajin Zhou, Haoyu Wang, and Kui Ren. 2021. Towards a first step to unimpderstand flash loan and its applications in defi ecosystem. In *Proceedings of the Ninth International Workshop on Security in Blockchain and Cloud Computing*. Association for Computing Machinery, New York, NY, USA, 23–28.

[40] Entriken William, Shirley Dieter, Evans Jacob, and Sachs Nastassia. 2018. ERC721: Non-Fungible Token Standard. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md.

[41] Radomski Witek, Cooke Andrew, Castonguay Philippe, Therien James, Binet Eric, and Sandford Ronan. 2018. ERC1155: Multi Token Standard. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1155.md.

[42] G Wood. 2019. Ethereum Yellow Paper: a formal specification of Ethereum, a programmable blockchain. *Accessed on: Mar* 6 (2019).

[5]email: qingqingsunmail@gmail.com
[6]email: 0xr3tsina@gmail.com
[7]email: wdboou@gmail.com
[8]email: allen@gopluslabs.io

[43] Siwei Wu, Zhou Yu, Dabao Wang, Yajin Zhou, Lei Wu, Haoyu Wang, and Xingliang Yuan. 2023. DeFiRanger: Detecting DeFi Price Manipulation Attacks. *IEEE Transactions on Dependable and Secure Computing* (2023).
[44] Zhiying Wu, Jieli Liu, Jiajing Wu, Zibin Zheng, Xiapu Luo, and Ting Chen. 2023. Know Your Transactions: Real-time and Generic Transaction Semantic Representation on Blockchain & Web3 Ecosystem. In *Proceedings of the ACM Web Conference*. Association for Computing Machinery, New York, NY, USA, 1918–1927.
[45] Zhiying Wu, Jiajing Wu, Hui Zhang, Ziwei Li, Jiachi Chen, Zibin Zheng, Qing Xia, Gang Fan, and Yi Zhen. 2024. DAppFL: Just-in-Time Fault Localization for Decentralized Applications in Web3. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. Vienna, Austria, 137–148.
[46] Maoyi Xie, Ming Hu, Ziqiao Kong, Cen Zhang, Yebo Feng, Haijun Wang, Yue Xue, Hao Zhang, Ye Liu, and Yang Liu. 2024. DeFort: Automatic Detection and Analysis of Price Manipulation Attacks in DeFi Applications. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. Vienna, Austria, 402–414.
[47] Jiashuo Zhang, Jianbo Gao, Yue Li, Ziming Chen, Zhi Guan, and Zhong Chen. 2022. Xscope: Hunting for cross-chain bridge attacks. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. ACM, Rochester, MI, USA, 1–4.
[48] Mengya Zhang, Xiaokuan Zhang, Yinqian Zhang, and Zhiqiang Lin. 2020. {TXSPECTOR}: Uncovering attacks in ethereum from transactions. In *USENIX Security Symposium*. USENIX Association, Boston, USA, 2775–2792.
[49] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. 2021. On the just-in-time discovery of profit-generating transactions in defi protocols. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 919–936.
[50] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. 2023. Sok: Decentralized finance (defi) attacks. In *IEEE Symposium on Security and Privacy*. IEEE, San Francisco, CA, USA, 2444–2461.
[51] Shunfan Zhou, Malte Möser, Zhemin Yang, Ben Adida, Thorsten Holz, Jie Xiang, Steven Goldfeder, Yinzhi Cao, Martin Plattner, Xiaojun Qin, et al. 2020. An ever-evolving game: Evaluation of real-world attacks and defenses in ethereum ecosystem. In *USENIX Security Symposium*. USENIX Association, Boston, USA, 2793–2810.
[52] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*.

A Appendix

A.1 Transaction Graph Modeling

The transaction execution process contains two types of data, i.e., execution logic and execution results [51]. The execution logic, e.g., function calls and execution instructions, is manifested in the instruction sequence or source code triggered during the transaction execution process. The execution logic determines the computations that the business logic corresponding to the transaction needs to accomplish. In addition, the execution results are typically reflected in receipts and event logs triggered by the transaction. Inspired by the prior work [25], we introduce a graph modeling for transaction, which incorporates the transaction execution logic and execution results.

Figure 9 demonstrates an toy model of the transaction graph. Note that the heterogeneous transaction data related to the transaction execution, including execution logic and execution results, can be obtained from RPC interfaces (i.e., *debug_traceTransaction* and *eth_getTransactionReceipt*) provided by blockchain clients. We introduce how these data are manifested in the transaction graph as follows:

Execution logic. In the transaction graph, the execution logic is modeled as a dynamic control flow. Classical code modeling methods, e.g., Abstract Syntax Trees (AST) and Control Flow Graphs (CFG), are designed for static program analysis and encompass both

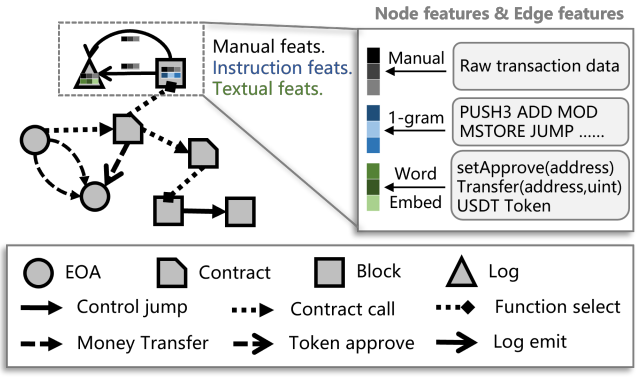


Figure 9: A toy model of the transaction graph.

executed and unexecuted execution logic. Consequently, these classical code modelling approaches may introduce noise in accurately understanding transaction execution logic due to the inclusion of a multitude of unexecuted instructions. To address this issue, we exclusively consider the instructions executed during the transaction execution. And we construct executed instructions into a dynamic control flow [25]. We design two types of nodes for dynamic control flow in the transaction graph, i.e., `Block` and `Contract`:

- **Block**: a basic block [29] in the transaction execution, each containing a set of executed instructions that terminate in a program branch.
- **Contract**: smart contracts that initiate function calls or are called during transaction execution.

Note that we use jump instructions (i.e., `JUMP` and `JUMPI`) and function call instructions (`CALL`, `CALLCODE`, `STATICALL`, `DELEGATECALL`, `CREATE`, `CREATE2`, and `SELFDESTRUCT`) to split the triggered instruction sequence into blocks. Moreover, we also design edges in the transaction graph for dynamic control flow:

- **Control jump**: the edge from `Block` to `Block`, denotes the control flow jumping between blocks.
- **Contract call**: the edge from `Contract` to `Contract`, denotes the program calling relationship between contracts.
- **Function select**: the edge from `Contract` to `Block`, denotes a specific block related to the smart contract function entry is selected.

Execution result. The execution result is also modeled in the transaction graph. Existing methods for modeling execution results, e.g., money transfer graphs [44], log sequences [38], etc., focus on modeling the relationship between execution results. However, current modeling approaches fail to establish contextual associations between execution logic and execution results. For instance, log sequences fail to provide insights into what instructions emit event logs during transaction execution. As the motivating example, the connection between event logs and code is crucial for the detection of on-chain attacks. As a result, we incorporate transaction execution results and dynamic control flow into the same transaction graph. Specifically, in the transaction graph, two types of nodes are related to execution results, i.e., `EOA` and `Log`:

- **EOA**: an externally owned account.

- **Log**: an event log emitted by the smart contract during the transaction execution.

And there are some edges related to transaction execution results:

- **Money transfer**: the edge between accounts (i.e., `EOA` and `Contract`), denotes the money transfer action defined by smart contract protocols (e.g., ERC20 [9], ERC721 [40], ERC1155 [41]) or blockchain systems.
- **Token approve**: the edge between accounts (including `EOA` and `Contract`), denotes the token approve action defined by smart contract protocols (e.g., ERC20, ERC721, ERC1155).
- **Log emit**: the edge from `Block` to `Log`, denotes the corresponding executed block emitting an event log.

Moreover, In the transaction graph, every node and edge has features. Prior research has demonstrated that conducting fine-grained feature modeling of blockchain transactions contributes to enhancing the effectiveness of learning-based models [38]. In this paper, the features within the transaction graph can be categorized into three types, i.e., manual features, instruction features, and textual features.

Manual features. In the transaction graph, manual features are extracted from raw transaction data through manually designed rules. For all nodes, manual features encompass both the out-degree and in-degree of the respective nodes. As for all edges, manual features include the index, i.e., the appearance order of the edge during transaction execution. And the index can be uniquely determined by the collected instruction sequence. Additionally, all data extracted from RPC interfaces that can be directly quantified is transformed into manual features.

Instruction features. The instruction feature exclusively manifest within `Block`. In fact, instructions in a `Block` cannot be directly quantified. To solve this problem, we employ the 1-gram technique [8], modeling the information contained in a set of instructions into features.

Textual features. Furthermore, we extract features from the text appearing in transactions. Specifically, text information within transactions primarily originates from log name, called function names, and asset names. Log names and function names can be reverse-engineered using topics and function signatures [7], respectively. We utilize reverse APIs provided by 4byte.directory[9] to decode log names and function signatures from raw transaction data into text. Additionally, asset names can be directly obtained through RPC interfaces, i.e., *eth_call*. Note that a transaction may include a lot of text, as multiple functions and event logs are triggered. And the majority of raw text in transactions is short phrases, e.g., "setApprove" and "transfer". Therefore, in order to trade off the efficiency and effectiveness of extracting text features, we adopted the classic text feature extraction method, i.e., the word average model [27]. Specifically, we tokenize phrases, followed by the extraction of word vectors for each word using pre-trained word embeddings [10]. After obtaining embeddings for each word, we average the word embeddings for all words in a given phrase, thereby generating a feature representing the phrase. The text features extracted from phrases are correspondingly added to the node and edge features in the transaction graph.

[9]https://www.4byte.directory/

```
/**
* @dev Performs a swap
* @param aggregatorId Identifier of the aggregator to be used for the swap
* @param data Dynamic data which is concatenated with the fixed aggregator's
* data in the delecatecall made to the adapter
*/
```

Figure 10: An example of the function comment.

A.2 Discussion of Comments

In the experiments, we collect the function comments for pre-training. After removing some unnecessary comment marks like "*" and "/", we have conducted some brief analyses on the comment data. The following are some of the conclusions:

- The total number of tokens in comments is 34.5 million.
- For tokens per comment, the average is 34.5, the maximum is 761, and the minimum is 4.
- 25% of the comments contain 4 to 13 tokens, 50% contain 13 to 34 tokens, and the remaining 25% of the comments exceed 34 tokens.

Especially, as comments serve as necessary supervision information, we ensure that all transactions have corresponding comments.

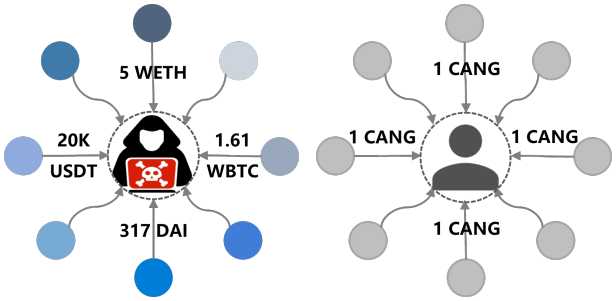
In fact, the information expressed in the comments is often condensed, which is not possessed by the transaction graph. For example, in the transaction graph, a transaction[10] contains dozens of nodes. Some complex high-order structures in the graph express semantics, but it may be hard to learn by the transaction reconstruction task, for it has to explore all possible high-order structures. On the contrary, the corresponding comment indicates the semantics (i.e., swap) directly. Therefore, the model pre-training can benefit from the comments, leading the model to learn meaningful transaction embeddings.

A.3 Discussion of Experiments in the Wild

A.3.1 Quality of Manual Verification. We use Fleiss' Kappa to evaluate the quality of manual verification. Fleiss' Kappa is a coefficient to measure inter-rater reliability for qualitative data [51]. For the result of our expert verification, the value of Fleiss' Kappa is large than 0.9. In general, a Fleiss' Kappa value greater than 0.8 indicates a high level of consistency in qualitative data. Therefore, the result of Fleiss' Kappa implies that the aforementioned manual verification results are consistent.

A.3.2 False Positive Analysis. Among the identified suspicious transactions, 1,054 non-attack transactions are included. Specifically, confirmed non-attack transactions include four categories, i.e., batch transfer (19.92%), mining (14.74%), arbitrage (0.54%), airdrop (0.22%), and others (2.75%). Batch transfer is the act of transferring multiple existing tokens in a single transaction, supporting by the batch transfer application, e.g., Cointool[11], disperse.app[12], etc. Mining distributes profits from a mining pool or token project to multiple accounts in a transaction. Arbitrage is commonly linked to

[10]0xd8a5cc098e58bb00bd89242401a3f4eec06c68c6b5a5a03aeb4160c23ca72f18
[11]https://cointool.app
[12]https://disperse.app/



(a) The Socket attack transaction. (b) A batch transfer transaction.

Figure 11: Comparison of token transfers between the Socket attack transaction and a non-attack transaction.

MEV bots [37, 50], which aim to exploit profit opportunities in various token swap services. And airdrop aims to distribute tokens to early project participants. Note that some non-attack transactions are difficult to fit into aforementioned categories. For example, one involves revenue management business, while performing complex operations, e.g., token deposit and token swap. Therefore, we classify this type of non-attack transaction as others.

For a deep-in analysis, we found that the proposed method may not be able to differentiate between attack transactions and non-attack transactions without training samples, as they share similar characteristics. For the sake of demonstration, we illustrate token transfers in the Socket attack (Figure 11(a)) and a batch transfer transaction (Figure 11(b)). Both consist of a hub account that acts as a token recipient and repeatedly participates in token transfers.

A.3.3 Defenses. In practice, to address the issue of false positives, we propose some suggestion for defenses:

(1) Increasing the size of the pre-training dataset. As discussed in the experimental section, randomly sampling transactions may miss some non-attack transactions with specific purposes, such as batch transfers. Therefore, one possible improvement is to expand the range of pre-training data, ensuring that the model learns a sufficiently diverse set of non-attack transactions, during the pre-training phase.
(2) Providing a small number of labeled samples. In the experiments corresponding to **Q1**, we find that just a few known samples can significantly improve transaction classification performance. Thus, by extracting a small number of FP samples and conducting attack transaction detection similar to Figure 5(a), it is also possible to effectively reduce the number of FPs.