23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

56

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

Anonymous Author(s)*

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://anonymous.4open.science/r/6F40.

CCS Concepts

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

Keywords

Attack detection, Blockchain transaction analysis, Web3

ACM Reference Format:

Anonymous Author(s). 2025. Hunting in the Dark Forest: A Pre-trained Model for On-chain Attack Transaction Detection in Web3. In *Proceedings* of Proceedings of the ACM Web Conference 2025 (WWW '25). ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXXXXXXXXXXXX

55 WWW '25, 28 April - 2 May 2025, Sydney, Australia

57 https://doi.org/XXXXXXXXXXXXXX



59 60

61 62

63 64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

Figure 1: The changes in the proportions among several onchain attack types over time.

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 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

^{© 2025} Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-XXXX-X/18/06

⁵⁸

attack taxonomies [6, 26, 50], and these attacks resulted in losses, 117 accounting for 52.5% of the total for the year [4]. According to an-118 other recent survey, only about 50% of new types of attacks [50] can 119 be detected by current tools. On the one hand, existing techniques 120 that rely on particular expert patterns may experience a gradual 121 decline in effectiveness, as the attack vector continues to evolve. 123 Especially when a new attack pattern appears, existing patterns 124 may not be able to cover new attacks. On the other hand, while 125 current learning-based approaches can extract attack knowledge 126 automatically, their efficacy is contingent upon well-defined optimization goals, which usually depend on a substantial volume of 127 128 labelled data or a clear attack taxonomy. Yet labeled on-chain attack samples and clear attack taxonomy are frequently absent for new 129 attacks, which hinders the efficacy of learning-based methods. 130

In the Web3 ecosystem with diverse and rapidly evolving on-131 chain attacks, it is an imperative but challenging task to design a 132 universal on-chain attack transaction detection model. First, it is 133 difficult to completely enumerate the patterns of on-chain attack 134 135 transactions. Blockchain transactions contain heterogeneous data. e.g., traces, logs, and receipts, and attack patterns are reflected in 136 137 the combination of these heterogeneous data [33, 38, 43]. Given that 138 transaction data encompass a variety of attributes and categories, 139 their combinations correspond to a space that is impossible to traverse within a limited time. Second, labeled on-chain attack 140 samples and a clear attack taxonomy are frequently absent for new 141 142 on-chain attacks. While some work establishes taxonomies and labels samples for new attacks [47, 50], these efforts are frequently 143 implemented subsequent to the occurrence of attacks. Prior to 144 145 the occurrence of new attacks, it is exceedingly challenging for condensing attack features that have not yet materialized. And thus 146 learning-based methods also struggle to learn generalizable decision 147 148 boundaries for on-chain attack transaction detection directly.

149 This paper aims to address the aforementioned issues by propos-150 ing a generic, learning-based method for detecting on-chain attack 151 transactions. First, we collect on-chain attack transactions from 152 2023 and earlier. Our analysis and prior work reveal that, at any given time, attack transactions exhibit some features (e.g., token 153 leakage [32, 45], price slippage [43, 46]) that significantly distin-154 155 guish them from non-attack transactions. And these features rarely appear in non-attack transactions. Based on the observation, we 156 offer a new insight for designing on-chain attack transaction de-157 tection models: implementing pre-training to learn the features of 158 159 reconstructing non-attack transactions. The pre-trained model can identify attack transactions using reconstruction error, without the 160 need to pre-defined specific attack patterns to be detected. More-161 162 over, we find that the generalizability of the pre-trained model can 163 be limited by the potential parameter over-fitting, resulting from 164 the reconstruction task. To prevent the over-fitting, the comment 165 corresponding to functions triggered during transaction execution is used as supervision. This allows the model to learn meaningful 166 embeddings in the reconstruction task. In this way, a generaliz-167 168 able on-chain attack transaction detection model can be developed, without any labeled attack transactions. 169

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.,

Anon.

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

reentrancy [33, 48], the proposed approach outperforms state-ofthe-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 transaction 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

233

251

266

267

268

269

270

271

273

274

275

276

277

278

279

280

281

290

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 237 238 aim to design specific patterns to detect on-chain attack transactions [37, 39, 43, 46, 48, 51]. TXSPECTOR [48] constructs triggered 239 240 the operation code into a graph, and uses the domain specific lan-241 guage to detect reentrancy, unchecked call, and suicidal attacks in 242 transacitons. Zhou et al. [51] modeled the transaction execution as 243 a graph, and defined six types of graph patterns to discover attack 244 transactions, e.g., reentrancy, honeypot, etc. Wang et al. [39] sum-245 marized a flashloan pattern for each provider to identify flashloan 246 transactions. Torres et al. [37] measured the three different types 247 of frontrunning attacks: displacement, insertion, and suppression, 248 based on their proposed heuristics. However, due to their design 249 for specific tasks, pattern-based methods may lack generalizability 250 and thus be difficult to scale to newly emerging attacks.

2.2.2 Learning-based Detection. Some learning-based methods [1, 252 19, 24, 33, 38, 44, 47] also attempted to extract transaction features 253 and discover attack behaviors in transactions. DeFiScanner [38] de-254 signed a neural network to fuse features from external transactions 255 and emitted events, which was then used to detect on-chain attacks. 256 DEFIER [33] modeled the transaction execution process as a graph, 257 and used graph embedding models and sequence embedding mod-258 els to classify multi-stage attack transactions. Recent work [1, 19] 259 has been devoted to extracting MEV (maximal extractable value) 260 actions in transactions through deep learning methods. However, 261 learning-based methods usually require a well-labelled dataset or 262 the clear attack taxonomy for model training. Thus, it is difficult for 263 existing learning-based methods to handle the new attack outside 264 of the training data. 265

3 Problem Analysis

This section conducts an analysis of the on-chain attack transaction task. Specifically, we explore the characteristics of attack and nonattack 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-282 order structures reflecting complex intent are composed of a small 283 number of high-order structure combinations [44]. One reason 284 for this phenomenon may be the code reuse [34], where devel-285 opers tend to use packaged code for constructing new programs. 286 Additionally, due to the large volume of non-attack transactions, 287 288 learning-based models can effectively learn the features of non-289 attack transactions [38, 44].



Figure 2: The frequency distribution of program segments. The coordinate axis is on a logarithmic scale.

Human experts are able to identify and report new attacks. In fact, the high-order structures corresponding to attacks rarely appear in non-attack transactions (e.g., token leakage [32, 45], price slippage [43, 46]). 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

WWW '25, 28 April - 2 May 2025, Sydney, Australia



Figure 3: The pre-training framework. Two tasks, i.e., transaction reconstruction (TXR) and transaction-comment contrast (TCC), are designed to train the model jointly.



Figure 4: The framework of PreTS.

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.

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 designed projections that maps the 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 structures 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 ∈ N₊ graph transformer blocks for feature propagation. Let G(·; θ⁽ⁱ⁾_g) denotes *i*-th graph transformer layer parameterized by θ⁽ⁱ⁾_g. The output of the graph transformer in the *i*-th graph transformer block is:

$$\hat{\mathbf{H}}_{n}^{(i+1)} = G(\mathbf{H}_{n}^{(i)}, \mathbf{H}_{e}, \mathcal{E}; \theta_{q}^{(i)}), \tag{4}$$

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

Anon

 $\mathbf{H}_e = \{\bigcup_t \mathbf{H}_t | t \in \Gamma_e\}$ is edge features. Especially, for the 1-th graph transformer, the input node features are:

$$\mathbf{H}_{n}^{(1)} = \left\{ \bigcup_{t} \mathbf{H}_{t} | t \in \Gamma_{n} \right\}.$$
(5)

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

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

(3) Pooling: We perform the average, mean, and max pooling operation on H_n^(l+1). Moreover, we concatenate all pooling output as the transaction features z_{tx}, i.e.,

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

where \parallel means concatenation, *Avg*, *Sum*, and *Max* denote the operation for calculating the average, mean, and maximum of all dimensions in $H_n^{(l+1)}$, 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{z}_{tx} is:

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

Let $T_i = E(\mathbf{z}_{tx}^i; \theta_e)$ denotes the embedding for the *i*-th transaction features \mathbf{z}_{tx}^i , and D_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.:

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

where $sim(T_i, D_j)$ means the cosine similarity between *i*-th transaction embedding T_i and *j*-th comment embedding D_j . And \mathcal{L}_{D2T}



(a) Detection with known samples (b) De

(b) Detection without known samples

Figure 5: Use the pre-trained transaction representation learning model, to perform on-chain attack detection with or without known attack samples.

makes the document embedding closer to the corresponding transaction embedding, i.e.:

$$\mathcal{L}_{D2T} = -\frac{1}{k} \sum_{i=1}^{k} \log\left(\frac{\exp(sim(D_i, T_i)/\tau)}{\sum_{j=1}^{k} \exp(sim(D_i, T_j)/\tau)}\right), \quad (12)$$

in which $sim(D_i, T_j)$ denotes the cosine similarity between the comment embedding D_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.

Experiments 5

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

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 BlockchainSpider¹, an RPC-based transaction data collection toolkit, to collect the heterogeneous transaction data.

610 5.1.2 Compared Methods. We select state-of-the-art approaches, 611 i.e., DeFiScanner [38], DEFIER [33], MoTS [44], and MetaSuites [3], 612 for comparison experiments. Note that all these selected methods 613 require a supervised training. Specifically, DeFiScanner fuses the 614 global and local features of a transaction and then processes the 615 fused features with LSTM [16] to train the model. DEFIER uses ver-616 tex embedding to work on the nodes in the transaction graph and 617 graph embedding methods to work on the transaction graph itself. 618 It then combines these two features and uses LSTM to work on them. 619 MoTS calculates the network motifs contained in the transaction 620 graph and uses the counting vector as the transaction represen-621 tation. Additionally, a Web3 transaction analysis tool, MetaSuites, 622 are taken into consideration. Based on ChatGPT, MetaSuites parses 623 transaction intent using data from blockchain browsers, e.g., Ether-624 scan², without requiring training samples. Note that the recent 625 proposed BlockGPT [11] has demonstrated its ability of detecting 626 on-chain attacks. Nevertheless, due to its under-review status and 627 the absence of reproducible details, BlockGPT is not utilized as a 628 comparative method in this paper. 629

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

637 ²https://etherscan.io

630

631

632

633

634

635



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.

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 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

6

Anon.

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

⁶³⁶ ¹https://github.com/wuzhy1ng/BlockchainSpider

⁶³⁸

755

756

771

772

773

774

775 776 777

778

779

780 781

782

783

784

785

786

787

788

789 790

791

792

793

794

795

796

797

798

799

800

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 transaction semantics. In the same supervision groups, the highest score is bolded, and the second highest score is underlined.

Supervised	Mathad	Reentrancy		Honeypot		Flashloan		Integer overflow		Call injection		AUC
Supervised	Methou	Р	R	Р	R	Р	R	Р	R	Р	R	
	DeFiScanner	1.00	0.83	0.00	0.00	0.85	0.32	1.00	0.65	0.67	1.00	0.78
E.,11	DEFIER	0.00	0.00	0.00	0.00	1.00	0.03	0.36	0.45	1.00	0.05	0.54
rull	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	<u>0.85</u>	<u>0.36</u>	<u>0.98</u>	0.70	<u>0.95</u>	<u>0.70</u>	0.94
	ChatGPT (MetaSuites)	0.00	0.00	0.00	0.00	1.00	0.15	0.00	0.00	0.00	0.00	0.515
Few/Zero-shot	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

Table 2: Variations in the efficacy of the proposed method as transaction reconstruction (TXR) or transaction-comment contrast (TCC) are eliminated. △F1-Score↓ indicates the decrease in macro F1 score.

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

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 shows the result of ablation study. Whether removing 743 744 transaction reconstruction or transaction-comment contrast, the performance of the proposed method decreases. Therefore, the pre-745 training does contribute to the efficacy of the proposed model in 746 on-chain attack detection. Note that when without transaction-747 748 comment contrast, the macro F1 score of the proposed method significantly decreases. For more explanation, we calculate the 749 distribution of reconstruction error for attack and non-attack trans-750 actions. Figure 7(a) shows that when the transaction-comment con-751 752 trast is used, the reconstruction error for the majority of non-attack 753 transactions is smaller than that for attack transactions. However,



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



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

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.

Detection in the Wild 5.4

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.

740

741

742

754

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

Category	Is attack?	# Transaction	(%)
Address poisoning	1	1,692	61.28%
Rug pull	1	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%

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 attack³ targeting the bussiness logic flaw in Socket project⁴, 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

⁴https://twitter.com/SocketDotTech

blockchain users [31]. Further analysis on the experiment in the wild can be found in Appendix A.2. 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.

6 Conclusion

In this work, we proposed a universal approach for detecting onchain 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 pretrained 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.

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:

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

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



Figure 9: A toy model of the transaction graph.

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 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 finegrained 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.directory⁵ 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

⁵https://www.4byte.directory/

WWW '25, 28 April - 2 May 2025, Sydney, Australia



Figure 10: Comparison of token transfers between the Socket attack transaction and a non-attack transaction.

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.

A.2 Discussion of Experiments in the Wild

A.2.1 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.2.2 False Positive Analysis. Among the identified suspicious 1078 1079 transactions, 1,054 non-attack transactions are included. Specif-1080 ically, confirmed non-attack transactions include four categories, 1081 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 trans-1082 ferring multiple existing tokens in a single transaction, supporting 1083 by the batch transfer application, e.g., Cointool⁶, disperse.app⁷, etc. 1084 Mining distributes profits from a mining pool or token project to 1085 1086 multiple accounts in a transaction. Arbitrage is commonly linked to 1087 MEV bots [37, 50], which aim to exploit profit opportunities in var-1088 ious token swap services. And airdrop aims to distribute tokens to early project participants. Note that some non-attack transactions 1089 are difficult to fit into aforementioned categories. For example, one 1090 involves revenue management business, while performing com-1091 1092 plex operations, e.g., token deposit and token swap. Therefore, we 1093 classify this type of non-attack transaction as others. 1094

For a deep-in analysis, we found that the proposed method may not be able to differentiate between attack transactions and nonattack transactions without training samples, as they share similar characteristics. For the sake of demonstration, we illustrate token transfers in the Socket attack (Figure 10(a)) and a batch transfer

1102

1095

1096

1097

1098

1099

1100

1058

1059

1060

1061

1062

1063

1064 1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

transaction (Figure 10(b)). Both consist of a hub account that acts as a token recipient and repeatedly participates in token transfers.

A.2.3 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.

References

- 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.
- 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. https://doi.org/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*

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

⁶https://cointool.app

^{1101 &}lt;sup>7</sup>https://disperse.app/

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

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1252

1253

1254

1255

on machine learning. ACM, Lille, France, 448-456.

1161

1218

- [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).
- [1172 [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.
- 1176 [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).
- 1179 [26] SWC Registry. 2020. Smart Contract Weakness Classification (SWC). https: 1180 //swcregistry.io/.
- 1181 [27] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084 (2019).
- [182 [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.
- 1188
 [30] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and

 1189
 Yu Sun. 2021. Masked Label Prediction: Unified Message Passing Model for

 1190
 Semi-Supervised Classification. In Proceedings of the Thirtieth International Joint

 1190
 Conference on Artificial Intelligence, IJCAI-21. International Joint Conferences on

 1191
 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. https://doi.org/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
 1206 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.
 [216] Jan Englisher Williger Williger Workshop and Seche Nettering 2018. EBC7211
- [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).
- [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*.