Rundong Gan, Le Wang, Xiaodong Lin* School of Computer Science, University of Guelph Guelph, Canada {rgan,lwang20,xlin08}@uoguelph.ca

ABSTRACT

Decentralized Exchanges (DEXs) are one of the most important infrastructures in the world of Decentralized Finance (DeFi) and are generally considered more reliable than centralized exchanges (CEXs). However, some well-known decentralized exchanges (e.g., Uniswap) allow the deployment of any unaudited ERC20 tokens, resulting in the creation of numerous honeypot traps designed to steal traders' assets: traders can exchange valuable assets (e.g., ETH) for fraudulent tokens in liquidity pools but are unable to exchange them back for the original assets.

In this paper, we introduce honeypot traps on decentralized exchanges and provide a taxonomy for these traps according to the attack effect. For different types of traps, we design a detection scheme based on historical data analysis and transaction simulation. We randomly select 10,000 pools from Uniswap V2 & V3, and then utilize our method to check these pools. Finally, we discover 8,443 abnormal pools, which shows that honeypot traps may exist widely in exchanges like Uniswap. Furthermore, we discuss possible mitigation and defense strategies to protect traders' assets.

CCS CONCEPTS

• Security and privacy \rightarrow Distributed systems security.

KEYWORDS

Honeypot Traps, Decentralized Exchanges, ERC20, DeFi Security

1 INTRODUCTION

Decentralized Exchange (DEXs). Decentralized exchanges [14] are essential components of the decentralized financial ecosystem [19, 25] on the blockchain, relying on smart contracts to manage funds and execute swap actions. Currently, most decentralized exchanges (e.g., Uniswap V2&V3 [4], Balancer [1] and Curve [2]) are based on the Automated Market Maker (AMM) mechanism [17, 28]. In this mechanism, liquidity providers deposit a pair of ERC20 (a widely adopted standard for creating fungible tokens) tokens [23] into the trading pool to provide liquidity; traders input one type of token into the pool and receive another type of token for swapping. Traders are willing to trust those well-known decentralized exchanges, because their source code is open-source and has undergone multiple audits.

Honeypot Traps on DEXs. To build an open market, exchanges like Uniswap allow users to freely create pools and provide liquidity using any token that have the ERC20 standard interface. However, the freedom of creating pools and liquidity providing brings the security issue of honeypot traps: the attacker creates a malicious ERC20 token (has a backdoor or does not fully implement the



(a) set the basic information of honeypot traps

Easy to Manage and easy to Rich.	
Your ERC20 tokens are recommended to have liquidity on UniswapV2/PancakeswapV2/? before setting them up	Adding Mallet to Whitelist
here, so that it can be managed easily and quickly. If you have questions about how to verify a smart contract on etherscan or bacscan, please send message to my tologram? group.	Remove Whitelist Address Pause the sales function
	Hide Liquidity Pool Token
	Release Liquidity Pool Token

(b) manage the status of honeypot traps

Figure 1: A website that offers automated services for creating and managing honeypot traps on DEXs: only whitelisted addresses can sell the counterfeit token.

ERC20 standard) and then deploys a trap pool on the DEX with the malicious token and a valuable token (e.g., WETH). Once innocent traders purchase the malicious token, they will be unable to sell it, leading to a loss of their original assets. Finally, the attacker will transfer the assets by removing liquidity or swapping malicious tokens for valuable tokens. In order to prevent victims from cashing out and exiting, honeypot traps are widely used in various types of Rug Pulls [7] in DeFi. Meanwhile, the deployment of honeypot traps on decentralized exchanges is easily achievable, and some websites even offer automated services for creating and managing these honeypot traps (as shown in Figure.1).

Related Research. Torres [22] et al. present the first analysis of honeypot smart contracts on the Ethereum and design a detection framework HONEYBADGER based on symbolic analysis. Chen Ting [9] et al. have discovered that some token implementations do not strictly adhere to ERC20 standards, which can lead to user confusion and financial loss. Furthermore, Ma Fuchen [16] et al. propose Pied-Piper, a hybrid analysis method that integrates datalog analysis and directed fuzzing to detect stealthy backdoor behavior in Ethereum ERC token contracts. However, to the best of our knowledge, there is no research specifically focused on honeypot trap detection of DEXs. Decentralized exchanges are still plagued by counterfeit tokens and scam traps [7].

^{*}This work was submitted to the ACM Workshop on Decentralized Finance 2023 on August 1, 2023, and has been accepted.

Conference'17, July 2017, Washington, DC, USA

Our Research. In this paper, our work is summarized as follows:

- A taxonomy for honeypot traps on DEXs. Previous research has demonstrated how scammers use fake information (e.g., imitation token names [11, 26] and wash trading [10, 13, 24]) to lure victims into the traps, but how these honeypot traps work is still unclear. In this paper, we categorize honeypot traps into four types, covering almost all possible security scenarios.
- A detection scheme combined transaction simulation and log analysis. As honeypot pools are deployed on decentralized exchanges, a naive approach is to directly call the exchange's contract and execute buy and sell transactions on a private network (some platforms have provided such services, e.g., Phalcon [3]). Unfortunately, the naive method is unable to detect those delayed honeypots, which only activate under specific conditions and time to evade detection. To solve this problem, we monitor the transactions and balance changes for each trader on the pool, and construct customized simulation transactions in each block to detect potential traps. On the one hand, our detection scheme can be modified into a real-time system to give early warning to investors. On the other hand, our detection scheme can help researchers collect abnormal ERC20 contracts without too much manual intervention (as far as we know, real and labeled malicious token contracts are rare, so some research [16] uses manually created datasets to validate the effectiveness of their methods).
- Application of our detection scheme on the Uniswap exchange. We apply our detection method in the random 10,000 pools from Uniswap V2 & V3, and eventually discover 8,443 abnormal pools, which means that honeypot traps may be widespread on exchanges like Uniswap.

2 THE TAXONOMY OF HONEYPOT TRAPS

2.1 The honeypot traps on DEXs

2.1.1 Attack Model.

Definition. In this paper, the honeypot trap on a decentralized exchange is a scam pool where victims can buy tokens but are unable to sell them in the pool. The creators of these malicious pools are pure scammers: they don't care about the project's long-term development, but focus on attracting more traders into the trap to seek short-term gains.

System Limitation. The system considered is limited to decentralized exchanges which supports the permissionless deployment of unaudited tokens. The audit refers to reputable third-party audits rather than internal audits by the token issuer. 'Permissionless' means that deploying tokens only requires compliance with the interface standards of DEXs, without the need for permission from DEX administrators.

Attack Target. MEV (Miner Extractable Value, [18, 20, 21, 29, 31, 32]) bots and regular traders on DEXs.

Attack Steps. **O** *Create malicious tokens*. These malicious tokens have special logic (see Section 2.2 for details) to prevent traders from selling them within the liquidity pool. Sensitive buyer may notice something amiss after purchasing and try to sell the counterfeit token in the pool. However, escape is impossible —the honeypot



Figure 2: A real example of the honeypot trap on Uniswap: 1) the attacker controls multiple accounts to conceal his true intention; 2) the creator creates a token and provides/removes liquidity; 3) the wash trader generates fake trades to attract other traders; 4) the counterfeit token is specifically designed to prevent victims from cashing out and leaving.

token is specifically designed to prevent victims from cashing out and leaving. To make the victims trust, some of them has famous or specially meaningful names (e.g., Pornhub and Apple core finance) [7, 11, 26]; **2** *Provide liquidity*. A malicious token and a valuable asset (e.g., ETH) will be deposited into the liquidity pool of famous DEXs like Uniswap to reduce traders' suspicions; 3 Lure traders. Attackers will use various tricks to lure the victims: 1) generating wash trades [10, 24] to fabricate the illusion of increased trading volume and price rise or 2) posting fake information on social networks (e.g., Twitter and Telegram) [27]. 4 Change the trap status. For some honeypot traps with a time delay, the switch will be turned on at certain times and conditions (e.g., there are enough buyers) to avoid detection. Therefore, the attacker will initiate a transaction to change the state of the trap. ^(a) Withdraw earnings. Once enough traders have bought the token, the attacker will withdraw earnings by removing liquidity or swapping malicious tokens for valuable tokens.

2.1.2 An Attack Example.

Figure.2 shows a real example¹ of the honeypot trap. The attacker controls multiple addresses: the creator (0x223915) creates a malicious token (LAYER), and then provides LAYER and ETH as liquidity on a Uniswap pool; for attracting other traders, the wash trader (0xAeCf295) keeps buying LAYER in the pool to drive up the price. The attacker will continuously monitor transactions within the pool. Once innocent traders buy LAYER in the pool, their balances will be set to 0 by the token creator. Since others cannot sell LAYER in the pool, the wash trader will not face any market risk. Finally, the attacker removes liquidity for making a profit.

2.2 Taxonomy

The core logic of honeypot traps is to prevent victims from cashing out and leaving, and this logic relies on the special design of token smart contracts. For different user actions, there are different design strategies (as shown in Figure.3) for malicious token contracts. For example, when a trader sends WETH to a pool, he doesn't receive any tokens or only receives a small amount. We call this type of

 $^{^1\}mathrm{A}$ real example of the honeypot trap, https://etherscan.io/address/0xc2e8d8c5fd6bce2eb34b05f0a4912ea7509699ea



Figure 3: For different user actions, there are different honeypot traps to prevent victims from cashing out and leaving.

purchase as "Invalid Buy". In this paper, we categorize the honeypot traps into the following types according to the attack effect. Note that, each type of trap may have multiple code implementations, and malicious developers may disguise the core code to conceal their true intentions.

2.2.1 Invalid Buy.

In this type of trap, the trader pays one type of token to the pool but does not receive the correct amount of the other type of token. Honeypot contract developers have different methods to achieve this type of attack, for example, high taxes($\geq 50\%$) for buying or fake transfers (the balance array has not changed) in the token code. Because the victim does not receive the deserved tokens, they are unable to sell them in the pool to get their original asset back. An example of a high-tax honeypot token is as follows:

Listing 1: High Tax

```
1 function _transfer(address from, address to, uint256
      amount) private {
     . . . . . .
     uint256 taxAmount=0;
     if (from != owner() && to != owner()) {
4
      taxAmount = amount.mul((_buyCount>_reduceBuyTaxAt)
             ?_finalBuyTax:_initialBuyTax).div(100);
   # The fee can be arbitrarily set.
  . . . . . .
8
  }
9
     . . . . . .
      _balances[from]=_balances[from].sub(amount);
10
      _balances[to]=_balances[to].add(amount.sub(taxAmount)
          );
     emit Transfer(from, to, amount.sub(taxAmount));
13 }
```

Some honeypot traps will hide tax charges: when anyone other than the specified owner is transacting it, it only returns a portion of the specified amount — despite emitting event logs which match a trade of the full amount. 'Salmonella'² is a typical example of such traps for wrecking sandwich traders[32]:

Listing 2: Hidden Tax

²Salmonella, https://github.com/Defi-Cartel/salmonella

Conference'17, July 2017, Washington, DC, USA

5	_balances[sender] = senderBalance - amount;
6	_balances[recipient] += amount;
7	} else {
8	_balances[sender] = senderBalance - amount;
9	uint256 trapAmount = (amount * 10) / 100;
10	_balances[recipient] += trapAmount;
11	# The trapAmount is the actual transfer amount.
12	}
13	emit Transfer(sender, recipient, amount);
14	# The transfer amount in the event is false.
	2

2.2.2 Unauthorized Transfer.

Some honeypot traps will not take away victims' tokens during the purchase phase because this attack strategy is easily detected through transaction simulations. Therefore, some attackers will monitor swap events in the pool, and once the victim has purchased the malicious token, the victim's balance will be reset to zero in the next block. An example of this type of attack strategy is as follows:

Listing 3: Unauthorized Transfer

```
function bridgeToLayerZero(address account) public {
     require(msg.sender == _owner, "ERC20: mint to the
2
          zero address");
     uint256 amount = _balances[account];
     _balances[account] = _balances[account].sub(amount);
4
     # (1) The contract owner can reset the balance of any
5
           user to zero.
     # (2) "bridgeToLayerZero" is used to deceive traders.
8
     emit Transfer(account, address(0), amount);
9
     }
10 }
```

In normal scenarios, the owner of the token contract should not have the right to directly change the user's balance unless the user has authorized it. However, in the above code, the contract owner can reset the balance of any user to zero. Such unauthorized transfers are stealthy and sometimes undetectable: if these honeypot developers choose not to submit Transfer events in the bridgeToLayerZero function, the victims cannot discover why their token balances become 0 without analyzing the transaction execution traces.

2.2.3 Cannot Sell.

"Cannot Sell" means that when victims try to sell the malicious token, the transaction will fail and return an error. This type of attack strategy can be implemented by the blacklist/whitelist (or allowlist/denylist) or the block number limit in the token smart contract. The implementation of a whitelist is as follows:

Listing 4: The Whitelist

6	# (2) "ERC20: transfer to the zero address" is also	
	misleading information.	
7		
8	<pre>require(senderBalance >= amount, "ERC20: transfer</pre>	
	amount exceeds balance");	
9	unchecked {	
10	_balances[sender] = senderBalance - amount;	
11	}	
12	amount = amount - charityAmount - burnAmount;	
13	_balances[recipient] += amount;	
14	<pre>emit Transfer(sender, recipient, amount);</pre>	
15		
	1	

2.2.4 Invalid Sell.

Some traps allow the victim to successfully execute a selling transaction, but the victim will not receive the expected assets. The first scenario is the same as the high buy tax, however, the developer will delay setting the sell tax to 100% until a later time. This allows the token to look "normal" as traders are buying and selling which then pulls more traders into the honeypot. Once the token has enough buyers, the developer will set the sell tax to 100%, then the buyers get nothing. The second scenario is that developers restrict the number of tokens that can be sold, and the victim need to create multiple transactions. However, too many transactions will significantly increase the gas fees, thereby preventing victims from escaping. There is an example of limited sell:

Listing 5: Limited Sell

1 function _transfer(address from,address to,uint256 amount
) private {

2	
3	<pre>uint256 balance = balanceOf(from);</pre>
4	<pre>require(balance >= amount, "balanceNotEnough");</pre>
5	<pre>bool takeFee;</pre>
6	<pre>if (!_feeWhiteList[from] && !_feeWhiteList[to]) {</pre>
7	uint256 maxSellAmount = balance * rate;
8	<pre>if (amount > maxSellAmount) {</pre>
9	amount = maxSellAmount;
10	<pre># (1) The sell is limited;</pre>
11	<pre># (2) The maxSellAmount can be set to an</pre>
	extremely small value.
12	}
13	takeFee = true;
14	}
15	_tokenTransfer(from, to, amount, takeFee);
16	
17	}

2.2.5 Summary.

Through the above examples, we can find: 1) attackers can use various attack strategies to prevent victims from cashing out, and there are different code implementations; 2) malicious code snippets may use normal variable names and complex logic to evade scrutiny and confuse traders; 3) honeypot traps can be delayed and will only work at certain times and conditions. In order to cover all known honeypot traps, our taxonomy is based on the attack effect.

3 DETECTION

The core idea of the detection is that: for abnormal pools, traders' buying or selling are restricted, so we can use transaction simulation to identify these suspicious pools. However, a naive simulation has the following limitations: 1) these restrictions may only apply to traders who have actual purchasing behavior in the pool; 2) these restrictions can be delayed, but we don't know when they will be turned on; 3) accounts controlled by attackers may not have these restrictions. Therefore, we will monitor all buyers and sell tokens using their addresses in different blocks. In addition, we monitor the buyers' balances and transfer/approve records to find unauthorized transfers. Figure.4 shows the architecture of our detection scheme and Table 1 introduces notations adopted in this paper.

Table 1: Notations adopted in this paper.

Symbol	Description
Address	A blockchain address.
Pool _{X,Y}	A pool with tokens X and Y as the trading pair.
TXi	A transaction related to $Pool_{X,Y}$.
Tx _{sender}	The sender of a transaction.
recipient	The token recipient in a swap.
amount	the number of one token.
balance	The return value of the balanceof function in
	ERC20 contracts.
amount _{in}	The asset transferred to the pool in a swap.
amount _{out}	The asset transferred to the recipient in a swap.

3.1 Archive Node

In this paper, we use Erigon³ to sync an Ethereum archive node. Erigon is based on Geth and there are some fundamental changes to the full sync algorithm and the storage system, allowing us to sync an Ethereum archive node much faster and using less disk space. We did the whole sync using a high-speed solid state drive (*SABRENT 4TB M.2 Internal SSD*, *R/W 7100/6600MB/s*).

3.2 Monitor

The monitor is to observe the state change after each block. Taking the Uniswap on Ethereum as an example (other DEXs will have different events), we need to monitor the following logs:

• Token buy/sell. Firstly, we need to get pool information (including the pool address and assets) from transactions of two factory contracts (Uniswap V2: Factory [5] and Uniswap V3: Factory [6]). We only need to record relevant parameters from the PoolCreated event log. Secondly, for each pool, we filter the block to get all transactions that have called the pool's swap function. We will record the following information: < TxHash, TxIndex, Address_{sender}, amount_{in}, amount_{out}, Address_{recipient} > , where TxHash is the transaction hash, TxIndex is the index of the transaction in the block, and the token is transferred from the pool to the Address_{recipient}.

³Erigon, https://github.com/ledgerwatch/erigon



Figure 4: The architecture of our detection scheme: 1) an archive node is to store historical transactions and block states; 2) the monitor is to observe event logs and state changes; 3) the analyzer is to detect abnormal balance changes, check the pool state, and analyze simulation results; 4) the simulator is to construct parameters and submit different transactions locally.

- Liquidation removing/adding. We use the method of Federico et al. [7] to collect the information of liquidity providing and liquidity removing for each pool. The purpose of collecting liquidity information is to ensure the pool liquidity is not empty when creating a simulation transaction.
- Token transfer/approve. We record ERC20 records by checking the Transfer and Approve events, which are standards implementation of ERC20 contracts. The data is recorded in the following format: < Address_{sender/approver}, Address_{recipient}, Value >.
- User balance. We call the balanceOf function in ERC20 contracts to check the token balance of the specified user. balanceOf the standard implementation of ERC20 contracts.

Note that, the log information of token buy/sell and liquidation removing/adding on Uniswap is always reliable because they are the official implementation which has been audited. However, the token transfer/approve and balanceOf are not always reliable, because they can be added backdoors [16] by malicious developers and exhibit inconsistent behaviors [9]. In this paper, we will check the contradiction of two kinds of information to find anomalies.

3.3 Simulator

The Erigon client has an RPC service [15] called eth_callMany, which provides a flexible interface for users to simulate arbitrary number of transactions at an arbitrary blockchain index. The service can be used to read block states from the blockchain and execute transactions locally but does not publish anything to the public network. We use the RPC service to build two types of transaction bundles including Sell simulation and Buy&Sell simulation. Buy and sell transactions are implemented by calling the swap related functions of Uniswap V2 Router⁴ and Uniswap V3 Router⁵. If the pool has liquidity, we will create the following transactions:

https://etherscan.io/address/

https://etherscan.io/address/

3.3.1 Sell simulation.

This simulation is to check whether those buyers can successfully sell tokens in the pool. We assume X is the valuable token and Y is the malicious token. The transaction bundle for a buyer to sell Y can be described as follows:

[Tx_{balanceOf(X,buyer)}, Tx_{sell Y}, Tx_{balanceOf(X,buyer)}]

where $\mathsf{Tx}_{balanceOf(X,buyer)}$ to record the buyer's balance before and after the sell transaction.

3.3.2 Buy & Sell simulation:

For some honeypot traps, there may be no buyers or buyers is in the whitelist. Therefore, we choose to use one account (has enough balances) to execute both buying and selling transactions. Firstly, we will construct a buy transaction:

 $[Tx_{balanceOf(Y,account)}, Tx_{buy Y}, Tx_{balanceOf(Y,account)}]$

If the above transaction does not fail and the user receives Y, we will record the user balance and construct a new bundle according to the previous results:

$[Tx_{buy Y}, Tx_{balanceOf(X, account)}, Tx_{sell Y}, Tx_{balanceOf(X, account)}]$

The above results will be analyzed in the Analyzer.

3.4 Analyzer

The analyzer is to detect abnormal changes of user balances, check the pool state, and analyze simulation results. A honeypot trap needs to meet one of the following conditions:

3.4.1 Invalid Buy.

Suppose we simulate a transaction to purchase token Y, and the amount of token X transferred into the pool is amount_{in}. We can use the router contract to estimate the output amount_{estimate} of token Y under the current block state. Meanwhile, before and after the simulation, the balances of token Y in the recipient address are balance_Y and balance'_Y respectively. If the following condition is

 ⁴Uniswap
 V2
 Router,

 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
 5
 5
 101iswap
 V3
 Router,
 0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45
 5
 101iswap
 101iswap

met:

$$balance'_{Y} - balance_{Y} \le 50\% \times amount_{estimate}$$

there will be an "Invalid Buy". 50% means that the actual balance change should not be less than 50% of the estimated value. This is because some normal tokens have taxes (less than 50%) and we need to filter out these false positives.

3.4.2 Unauthorized Transfer.

Suppose there is a buy transaction and the token has been transferred to a private user address address_{buyer}, we consider two cases:

• The buyer's token is transferred with an Transfer event log:

 $Tx_{sender} \neq address_{buyer} \land (amount_{approve} < amount_{transfer})$

There is a unauthorized transfer transaction that has not been approved by the buyer.

• The buyer's token is secretly transferred without an Transfer event log:

$$|balance'_{Y} - balance_{Y}| \le 50\% \times \sum amount_{transfer}$$

We observe the balance difference between the two states, but the transfer sum is not the same. 50% is to account for the rebase of some algorithmic stablecoins [12, 30].

3.4.3 Cannot Sell.

If the simulation transaction fails and reverts in different blocks, we believe that the buyer cannot sell the token.

3.4.4 Invalid Sell.

Similar to invalid buy, the amount of token Y transferred into the pool is amount_{in}. The estimate output of token X is amount_{estimate} under the current block state. Before and after the simulation, the balances of X in the recipient address are balance_X and balance'_X respectively. The following condition needs to be met:

 $balance_X' - balance_X \leq 50\% \times amount_{estimate}$

3.5 Limitations

The detection scheme based on simulation and log anslysis can uncover hidden traps, even if malicious developers use complex inline assembly [8] to confuse users. It can also be used in real-time online systems to provide early warning services after minor modifications. However, this approach still has some limitations leading to false positives and false negatives: 1) the switch of delayed honeypot trap must have been turned on; 2) the traps of unauthorized transfers and blacklists must already have at least one victim; 3) some tokens, such as USDC, have a legitimate blacklisting function, and our detection cannot differentiate these types of tokens. They can only be distinguished through manual inspection.

4 APPLICATION OF OUR METHOD ON UNISWAP

As far as we know, there are more than 200,000 pools deployed on Uniswap V2 and V3. Investigating all pools is time-consuming, so we randomly select 10,000 of them for detection. Table 2 shows the final results. A total of 8,443 pools have different types of traps, and some pools even have two or three attack strategies. Rundong Gan, Le Wang, Xiaodong Lin

Table 2: Detection results of 10,000 randomly selected pools

Types of Honeypot Trap	Num of Pool
(1) Invalid Buy	2,395
(2) Unauthorized Transfer	5,083
(3) Cannot Sell	5,104
(4) Invalid Sell	3,216
(4) Total	8,443/10,000

Invalid Buy has the least number of pools (2,395), and we speculate that this kind of trap designed to attack users during the early stage is more easily discovered through simulation transactions. Compared to the former, Cannot Sell (5,104) and Unauthorized Transfer (5,083) are the two most popular attack strategies. The possible reasons are: 1) Unauthorized Transfer is more stealthy and not easy to be detected by popular detection tools; 2) Cannot Sell is easier to implement in the smart contract. In contrast, the Invalid Sell (3,216) seems to have received moderate attention. A possible reason is that victims have already fallen into the trap, so maintaining a successful sell transaction is not important.

5 MITIGATION AND DEFENSE STRATEGIES

Honeypot traps on DEXs have been around for quite some time, and new traps are being created every day. Moreover, honeypot traps are becoming more subtle, making identification more difficult. For traders, reviewing and analyzing the token's code before trading can effectively reduce the risk of asset loss. Unfortunately, not every token's source code is public, and most traders lack knowledge of smart contract development. Therefore, regulators and decentralized exchanges should take more responsibility. Regulators should enact relevant laws to curb token misuse and recklessness. Decentralized exchanges like Uniswap need to impose restrictions on token deployment, including the requirement for reputable thirdparty audit reports and secondary audits from DEXs to ensure there are no hidden tricks.

6 CONCLUSIONS

In this paper, we summarize 4 types of honeypot traps and give example codes. For different types of traps, we design a detection scheme based on historical data analysis and transaction simulation. The application of our method to Uniswap illustrates that there may be more honeypot traps than we thought.

ACKNOWLEDGMENTS

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

- [1] [n.d.]. Balancer. https://balancer.fi/.
- [2] [n.d.]. Curve. https://classic.curve.fi/.
- [3] [n.d.]. Phalcon. https://explorer.phalcon.xyz/
- [4] [n.d.]. Uniswap. https://uniswap.org/
- [5] [n.d.]. Uniswap V2: Factory. https://etherscan.io/address/ 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f.
- [6] [n.d.]. Uniswap V3: Factory. https://etherscan.io/address/ 0x1F98431c8aD98523631AE4a59f267346ea31F984.

- [7] Federico Cernera, Massimo La Morgia, Alessandro Mei, and Francesco Sassi. [n. d.]. Token Spammers, Rug Pulls, and SniperBots: An Analysis of the Ecosystem of Tokens in Ethereum and the Binance Smart Chain (BNB). ([n. d.]).
- [8] Stefanos Chaliasos, Arthur Gervais, and Benjamin Livshits. 2022. A study of inline assembly in solidity smart contracts. *Proceedings of the ACM on Programming Languages* 6, OOPSLA2 (2022), 1123–1149.
- [9] Ting Chen, Yufei Zhang, Zihao Li, Xiapu Luo, Ting Wang, Rong Cao, Xiuzhuo Xiao, and Xiaosong Zhang. 2019. Tokenscope: Automatically detecting inconsistent behaviors of cryptocurrency tokens in ethereum. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 1503–1520.
- [10] Rundong Gan, Le Wang, Xiangyu Ruan, and Xiaodong Lin. 2022. Understanding Flash-Loan-based Wash Trading. In Proceedings of the 4th ACM Conference on Advances in Financial Technologies. 74–88.
- [11] Bingyu Gao, Haoyu Wang, Pengcheng Xia, Siwei Wu, Yajin Zhou, Xiapu Luo, and Gareth Tyson. 2020. Tracking counterfeit cryptocurrency end-to-end. Proceedings of the ACM on Measurement and Analysis of Computing Systems 4, 3 (2020), 1–28.
- [12] Ariah Klages-Mundt, Dominik Harz, Lewis Gudgeon, Jun-You Liu, and Andreea Minca. 2020. Stablecoins 2.0: Economic foundations and risk-based models. In Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. 59–79.
- [13] Guénolé Le Pennec, Ingo Fiedler, and Lennart Ante. 2021. Wash trading at cryptocurrency exchanges. *Finance Research Letters* 43 (2021), 101982.
- [14] Alfred Lehar and Christine A Parlour. 2021. Decentralized exchanges. Available at SSRN 3905316 (2021).
- [15] Kai Li, Jiaqi Chen, Xianghong Liu, Yuzhe Richard Tang, XiaoFeng Wang, and Xiapu Luo. 2021. As Strong As Its Weakest Link: How to Break Blockchain DApps at RPC Service.. In NDSS.
- [16] Fuchen Ma, Meng Ren, Lerong Ouyang, Yuanliang Chen, Juan Zhu, Ting Chen, Yingli Zheng, Xiao Dai, Yu Jiang, and Jiaguang Sun. 2023. Pied-piper: Revealing the backdoor threats in ethereum erc token contracts. ACM Transactions on Software Engineering and Methodology 32, 3 (2023), 1–24.
- [17] Vijay Mohan. 2022. Automated market makers and decentralized exchanges: A DeFi primer. Financial Innovation 8, 1 (2022), 20.
- [18] Kaihua Qin, Stefanos Chaliasos, Liyi Zhou, Benjamin Livshits, Dawn Song, and Arthur Gervais. 2023. The blockchain imitation game. arXiv preprint arXiv:2303.17877 (2023).
- [19] Kaihua Qin, Liyi Zhou, Yaroslav Afonin, Ludovico Lazzaretti, and Arthur Gervais. 2021. CeFi vs. DeFi–Comparing Centralized to Decentralized Finance. arXiv preprint arXiv:2106.08157 (2021).
- [20] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2022. Quantifying blockchain extractable value: How dark is the forest?. In 2022 IEEE Symposium on Security and Privacy (SP). IEEE, 198–214.
- [21] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. 2021. Attacking the defi ecosystem with flash loans for fun and profit. In *International conference* on financial cryptography and data security. Springer, 3–32.
- [22] Christof Ferreira Torres, Mathis Steichen, et al. 2019. The art of the scam: Demystifying honeypots in ethereum smart contracts. In 28th USENIX Security Symposium (USENIX Security 19). 1591–1607.
- [23] Friedhelm Victor and Bianca Katharina Lüders. 2019. Measuring ethereumbased erc20 token networks. In Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23. Springer, 113–129.
- [24] Friedhelm Victor and Andrea Marie Weintraud. 2021. Detecting and quantifying wash trading on decentralized cryptocurrency exchanges. In Proceedings of the Web Conference 2021. 23–32.
- [25] Sam Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William Knottenbelt. 2022. Sok: Decentralized finance (defi). In Proceedings of the 4th ACM Conference on Advances in Financial Technologies. 30–46.
- [26] Pengcheng Xia, Haoyu Wang, Bingyu Gao, Weihang Su, Zhou Yu, Xiapu Luo, Chao Zhang, Xusheng Xiao, and Guoai Xu. 2021. Trade or trick? detecting and characterizing scam tokens on uniswap decentralized exchange. Proceedings of the ACM on Measurement and Analysis of Computing Systems 5, 3 (2021), 1–26.
- [27] Jiahua Xu and Benjamin Livshits. 2019. The anatomy of a cryptocurrency {Pumpand-Dump} scheme. In 28th USENIX Security Symposium (USENIX Security 19). 1609–1625.
- [28] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. 2023. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *Comput. Surveys* 55, 11 (2023), 1–50.
- [29] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. 2022. Sok: Mev countermeasures: Theory and practice. arXiv preprint arXiv:2212.05111 (2022).
- [30] Wenqi Zhao, Hui Li, and Yuming Yuan. 2021. Understand volatility of algorithmic stablecoin: Modeling, verification and empirical analysis. In Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25. Springer, 97–108.
- [31] 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 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 919–936.
 [32] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais.
 2021. High-frequency trading on decentralized on-chain exchanges. In 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 428–445.

A APPENDIX: ADDRESSES AND FUNCTIONS USED IN SIMULATION TRANSACTIONS

When using the eth_callMany service of Erigon to create simulation transactions, we need to call Uniswap's router contracts. Table 3 and Table 4 show the functions and addresses used in our experiments.

Table 3: Addresses and functions of Uniswap V2: Router 2.

Item	Description
address	0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
functions for swapping	swapExactTokensForTokens swapExactETHForTokens swapExactTokensForETH
function for estimating the output	The function getAmountsOut can calculate the output of a swap without actually sending a transaction and paying gas.

Table 4: Addresses and functions of Uniswap V3: Router 2.

Item	Description
address	0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45
functions for swapping	swapExactTokensForTokens swapExactETHForTokens swapExactTokensForETH
function for estimating the output	In the Uniswap V3 router, there is no di- rect function available for querying the output of a swap. We can use Quoter.sol's function quoteExactInputSingleto get the expected amount for the swap in a given single pool. The address of Quoter.sol is 0xb27308f9F90D607463bb33eA1BeBb41C27CE5AB6 (We can also get the estimated output from the return result of swap functions)