Lost in Blockchain Address Misuse: Hidden Cross-Platform Risks and Their Security Impact

Zhenzhe Shao[1,2], Jiashuo Zhang[3], Zihao Li[4,5], Daoyuan Wu[6],
Chong Chen[1], Yiming Shen[1], Lingfeng Bao[2,7], Yanlin Wang[1], Jiachi Chen[2,7,*]
[1]*School of Software Engineering, Sun Yat-sen University*
[2]*The State Key Laboratory of Blockchain and Data Security, Zhejiang University*
[3]*Peking University,* [4]*University of Electronic Science and Technology of China*
[5]*The Hong Kong Polytechnic University,* [6]*Lingnan University*
[7]*Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security*

Abstract

Blockchain systems, such as Ethereum, employ an account-based model, where each account is uniquely identified by an address. As the fundamental interface for user interaction and asset security, addresses are critical but also pose significant risks when misused. In this paper, we systematically reveal and analyze a class of risks termed *Address Misuse*, which includes two categories: *Contract Account (CA) Misuse* and *Externally Owned Account (EOA) Misuse*. Specifically, CA Misuse arises when users mistakenly treat non-contract addresses (NCAs) as CAs, while EOA Misuse occurs when users interact with EOAs whose private keys are exposed. For each category, we reveal the underlying mechanisms and also introduce previously undisclosed attack vectors that enable attackers to exploit these vulnerabilities for profit. To evaluate their prevalence and impacts, we first construct a dataset from GitHub and Stack Exchange, which contains addresses of various blockchain networks. This dataset includes 10 million candidate addresses for misuse analysis and 16 million exposed private keys. We then perform a large-scale on-chain analysis of their associated transactions on Ethereum and BSC. By combining heuristic rules, transaction pattern analysis, and symbolic execution, we identify 65,340 high-risk address instances, with associated asset losses amounting to about 127k ETH and 17.7k BNB, equivalent to over $574.8M. We evaluate the accuracy of our detection methods to ensure the reliability of the results, achieving an overall precision of 99.11%. Besides, our empirical evaluation also reveals two novel, previously undisclosed attack vectors, providing real-world evidence of how attackers actively exploit users' address misuse for profit.

1 Introduction

Blockchain technology, with its decentralized, transparent, and immutable nature, has enabled a wide range of applications [63]. Most blockchains, such as Ethereum [10], adopt an account-based model, where each account represents a state entity on the ledger and is uniquely identified by an address. These addresses are the core interface for user interaction with blockchain systems, which serve as unique user identifiers, asset containers, and entry points for smart contract execution. Consequently, the correct and secure use of addresses is critical for ensuring asset protection and interaction validity.

Despite their importance, addresses also constitute a potential vector for security risks. Due to negligence, misoperation, or lack of knowledge, users may interact with unsafe or unintended addresses, even directly transferring tokens to these addresses. Such incorrect address interactions, collectively referred to as ***Address Misuses*** in this paper, have caused prevalent and high-volume loss of assets in the real world. To systematically understand and measure this risk, our work conducts the ***first large-scale measurement*** of address misuses. We classify address misuses into two types based on account categories: *Externally Owned Accounts (EOAs)* and *Contract Accounts (CAs)* [16]. EOAs are controlled by private keys and mainly used for asset transfer, while CAs are deployed smart contracts invoked through functions. Misuse in either case can cause irreversible loss, which highlights the need for a comprehensive measurement.

The first type, *Contract Account (CA) Misuse*, occurs when users mistakenly treat non-contract accounts (NCAs) as CAs. An NCA refers to an address without deployed contract code, which includes EOAs and not-yet-deployed contract addresses. This misuse often arises from address reuse or confusion across different blockchain networks. For example, on the Sepolia testnet [24], the address `0xC532...008` hosts an unofficial deployment of the `UniswapV2Router02` [58] contract. This address is widely referenced on developer Q&A platforms such as Stack Exchange [27–29] and is frequently used for testing purposes. However, on the Ethereum mainnet, the same address is an NCA. Many users mistakenly send function call transactions (even with ETH attached) to the mainnet addresses, under the false assumption that it corresponds to the same contract. As a result, the call is processed as a simple transfer, and any attached funds become trapped.

*Corresponding author.

The second type, *Externally Owned Account (EOA) Misuse*, arises when users interact with EOAs whose private keys are exposed. These exposed keys may originate from test accounts in public code repositories or examples on developer Q&A platforms. A typical example is the address `0x6273...Ef57`, a default account in the popular Solidity contract development suite Truffle [56]. This account is pregenerated for local testing, with its private key publicly available in Truffle's codebase [55]. However, some users unknowingly use this account in real-world blockchain transactions, unaware that it can be controlled by anyone. Consequently, any funds sent to this address can be transferred out immediately by anyone knowing the exposed private key.

To investigate the prevalence and security impact of real-world address misuses, we present an approach for the detection. Motivated by a manual examination of real-world misuse instances, we extract their behavioral features and transaction patterns. The designed detection framework combines transaction pattern analysis and symbolic execution. Specifically, our tool first traverses all transactions of candidate addresses and matches them against predefined misuse patterns. It further applies symbolic execution to examine whether the involved contract logic contains malicious transfers. Addresses that satisfy these rules are marked as misuse cases, and the associated asset losses are recorded.

In the evaluation, we first construct a dataset from GitHub and Stack Exchange, which contains over 10 million candidate addresses and 16 million exposed private keys across different blockchain networks for. Then, using our proposed detection approach, we analyze the transactions of these addresses on Ethereum mainnet [21] and BNB Smart Chain (BSC) [9] to identify misuse instances and quantify associated financial losses. To ensure the reliability of our evaluation, we manually validate our approach on a randomly sampled ground truth dataset, confirming that it achieves an overall precision of 99.11% in detecting address misuses.

The evaluation results demonstrate that both CA Misuse and EOA Misuse are widespread and result in significant financial losses. From the collected dataset, we identify 49,344 high-risk addresses related to CA Misuse with losses of 22,738.41 ETH and 8,681.41 BNB, and 15,996 high-risk addresses related to EOA Misuse with losses of 104,244.53 ETH and 9,045.29 BNB. In total, all these 65,340 misuse address instances involve 2.5M transactions, resulting in financial losses of approximately 127k ETH and 17.7k BNB, valued at over $574.8M. These findings highlight critical and unrecognized security weaknesses in both user behavior and development practices, demonstrating the prevalence and severity of Address Misuse.

Furthermore, our empirical evaluation reveals two new attack vectors where real-world attackers *actively* exploit users' address misuse to conduct profitable attacks. In CA Misuse, we find 469 cases where attackers leverage cross-chain address reuse by deploying contracts on testnets and later planting malicious contracts at the same addresses on mainnets to transfer trapped funds. These attack result in the loss of 3,446.37 ETH and 431.79 BNB. Among these, 89.76% of malicious contracts implement direct transfer logic, while 6.82% adopt proxy contract patterns. In EOA Misuse, we find 17,270 cases where attackers exploit the new EIP-7702 mechanism to delegate control of exposed EOAs to malicious contracts, allowing them to instantly drain incoming funds. This vector has led to losses of 25.86 ETH and 33.45 BNB, exhibiting highly patterned with the top three malicious contracts alone accounting for 78.71% of EIP-7702 delegations.

The main contributions of our work are as follows:

- We conduct the first systematic analysis of address misuses on blockchain and define two types of address misuses based on whether CAs or EOAs are involved.
- We present a tool that combines large-scale off-chain data mining with on-chain transaction and contract analysis to detect and measure real-world address misuse.
- Our empirical evaluation demonstrates the prevalence and security impact of real-world address misuse. The empirical evaluation identifies 49,344 CA Misuse and 15,996 EOA Misuse instances on Ethereum and BSC, with total financial losses reaching 127k ETH and 17.7k BNB, valued at $574.8M.
- Our empirical evaluation reveals two previously undisclosed attack vectors, providing real-world evidence of how attackers actively exploit address misuses for profit. We also provide practical recommendations for developers and users to mitigate address misuse.

2 Background

2.1 Blockchain Accounts and Addresses

In blockchain, an address is the unique identifier for a user or smart contract, which can be classified into two types [35]. (1) Externally Owned Addresses (EOAs), which are controlled by the private key holder and are the only entity that can sign and initiate transactions [4]. (2) Contract Addresses (CAs), which are controlled by a smart contract, not by a private key directly. CAs are deterministically generated during contract deployment. EOAs deploy contracts by sending a special transaction that invokes the CREATE opcode [3].

This study also defines another category called **Non-Contract Address (NCAs)**, which refers to addresses without contract code. NCAs include both EOAs and Potential Contract Addresses (PCAs). Since the address of a CA can be precomputed, any yet-unused address that could host a contract in the future is classified as a PCA. Similarly, a PCA does not have a corresponding private key to directly control it. Its control can only be claimed by deploying a contract.

2.2 Contract Address Calculation

Contracts are deployed when an EOA sends a transaction that invokes the `CREATE` opcode [45]. Such a transaction has an empty `to` field and a `data` field containing the contract's creation bytecode, which includes the compiled runtime bytecode and initialization code. Upon processing, the EVM executes the creation bytecode, initializes the contract's storage, and records its runtime bytecode at the newly created CA [37, 38]. The CA is computed as `keccak256(rlp_encode(deployer_address, nonce))`, where `nonce` is a transaction counter for the deploying address [50]. The nonce starts at `0` and increments by one with each transaction. Thus, the contract address calculation depends solely on the deployer's address and its nonce. Since nonces are maintained independently across different blockchains, a developer can deliberately adjust the nonce to deploy distinct contracts on multiple chains (e.g., Ethereum and BNB Chain) that share the same contract address.

2.3 Calling Smart Contract Functions

Users interact with smart contracts by sending transactions to their addresses [42]. The transaction's `to` field is the target contract address, and its input data field contains the Function Selector and the encoded Arguments. The function selector is the first four bytes of the Keccak256 hash of the function signature. For example, the function selector for `transfer(address,uint256)` is `0xa9059cbb`. Arguments encoding follows the ABI specification [17]. The EVM uses the input data to identify and execute the target function.

A critical but often overlooked mechanism is that when a transaction containing function call data is sent to a NCA, the transaction does not fail or revert but executes successfully. The EVM simply treats it as a regular transfer, ignoring the input data. Any attached native token, e.g., ETH and BNB, is still transferred to that address successfully. This is the core technical cause of financial loss in Address Misuse.

2.4 EIP-7702

EIP-7702 is a recently introduced Ethereum improvement proposal (EIP) in Pectra upgrade [20]. It allows EOAs to authorize a special type of transaction that delegates its execution rights to a designated smart contract [11]. This effectively transforms the EOA into a CA. The EOA's code field is set to `0xef0100 || address`, where `0xef0100` is a fixed identifier and address denotes the delegate contract's address. The code field remains until the EOA revokes the delegation. Although designed to enhance the flexibility of EOAs, EIP-7702 introduces new risks. It provides a more efficient mechanism for EOA Misuse attacks. An attacker who knows the exposed private key can use EIP-7702 to delegate the compromised EOA to a malicious contract, which is pre-designed and pre-deployed to sweep funds.

3 Address Misuses

3.1 Preliminaries

In our model, the blockchain system is composed of users, addresses, and transactions. Address types include EOAs, CAs, and NCAs. Users interact with the system by sending transactions to the addresses. These transactions aim to transfer assets or invoke contract functions. Within this system, benign users intend to correctly transfer assets or call contract functions, while malicious users monitor the blockchain to capture misused funds.

System model. We define Address Misuse as the scenario where a user interacts with an incorrect or unsafe address. This threat does not rely on the execution of malicious code. Instead, it originates from users' misunderstanding of address types or neglect of the blockchain's network context. Address Misuse occurs when a user sends a transaction under false assumptions about an address's type or security state, leading to irreversible fund loss. In this paper, we focus on CA Misuse and EOA Misuse, two primary categories of this threat.

3.2 CA Misuse

Normal interactions with CA. In a normal interaction with a smart contract, a user sends transactions to invoke a function on a CA, expecting the deterministic execution defined by its code. Such transactions are either executed correctly, or reverted to protect user assets from accidental loss.

CA Misuse. CA Misuse occurs when users mistakenly treat a non-contract address A_{NCA} as their target contract address C_{target} and sends a function call transaction tx_{func} with value v (e.g., ETH). Since A_{NCA} contains no executable code, EVM-based blockchains process tx_{func} as a successful simple transfer instead of reverting it. As a result, the attached value v is credited to the balance of A_{NCA} and becomes trapped permanently. The critical risk of CA Misuse lies in the silent success of the transaction: *users often do not realize the loss until they check the expected outcome.*

These misused NCAs may include EOAs, typically due to input errors, or PCAs that arises from user's confusion about address reusability across different networks. The core risk lies in the fact that the same blockchain addresses exhibit entirely different behaviors across networks; users often assume that a contract deployed on one chain also exists at the same address on another chain.

Example. We analyze a real-world example about `UniswapV2Router02` [58] contract on the Sepolia testnet [24] to demonstrate the process of CA Misuse, as shown in the User part of the green line segment box in Figure 1. The misused address, `0xC532...4008`, appears many times as an answer in Stack Exchange posts [27–29] with more than 102k views. On the Sepolia testnet, this address[1] is de-

[1] https://sepolia.etherscan.io/address/0xC532...4008

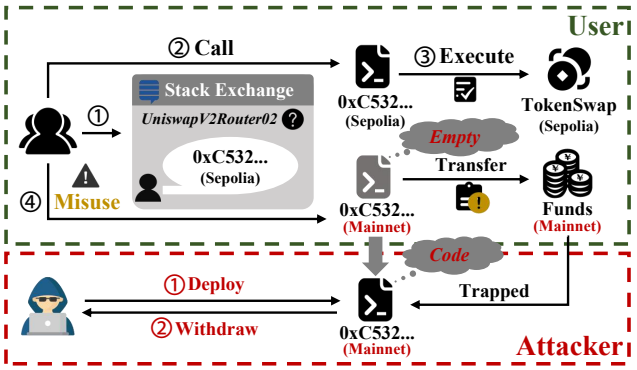

Figure 1: CA Misuse and Attack Process on Address `0xC532...4008` (Green line segment box: User-Induced CA Misuse; Red line segment box: Attacker Exploitation)

ployed with the `UniswapV2Router02` contract on May 9, 2023 (block height: 3,448,360). Users call functions like `swapETHForExactTokens`, `swapExactETHForTokens`, and `addLiquidityETH` for testing purposes. By the end of August 2025, this address has approximately 158,775 transactions on Sepolia testnet, indicating its high-frequency use.

However, on Ethereum mainnet, interactions with this same address[2] show the CA Misuse. Before October 2024 (mainnet block height: 20,902,977), the address remained a PCA with no contract code. Despite this, there were 88 transactions in which users attempted to initiate function calls to this address, such as transactions `0x2c2c`[3] and `0x76d2`[4]. These users intend to interact with the Uniswap V2 Router, calling the `swapETHForExactTokens` function and attaching ETH. Since there is no contract deployed at the address at that time, any function call to it is processed as a simple transfer. The transactions do not revert, and the attached ETH becomes trapped at the address.

3.3 EOA Misuse

Normal interactions with EOA. When transferring funds to an EOA, the user expects the recipient to be a trusted peer who exclusively controls the corresponding private key.

EOA Misuse. EOA Misuse occurs when a user sends a transaction tx_{in} transferring assets v to $EOA_{exposed}$ whose private key $key_{exposed}$ is publicly exposed. An exposed private key means that malicious users can easily obtain it and take full control of the corresponding address and its assets. They often use automated programs such as MEV bots to continuously monitor $EOA_{exposed}$. Upon detecting the incoming transfer tx_{in}, they immediately use $key_{exposed}$ to create, sign, and broadcast an outgoing transaction tx_{out} to steal the funds. Any funds arriving at $EOA_{exposed}$ are instantly drained by the fastest malicious actors. The victim user's loss is immediate and irreversible.

In practice, these private key exposures originate from various sources, including: *(1) Public Code Repositories.* Code hosting platforms like GitHub are a common source for private key exposures. For example, some blockchain development frameworks such as Truffle [56] or Hardhat [43] provide a default set of test accounts with hard-coded private keys in their documentation or source code to facilitate local testing. Users may mistakenly believe these accounts are randomly and securely generated by the tool and use them in mainnet transactions. Additionally, some developers unintentionally commit configuration files or scripts containing private keys when uploading projects, resulting in exposure. *(2) Online Q&A Platforms.* Developers sometimes provide example private keys and addresses when responding to questions on platforms such as Stack Exchange. These addresses may be default public test accounts or attacker-controlled addresses. Inexperienced users may copy and reuse them directly in their projects, leading to mistaken interactions on the mainnet.

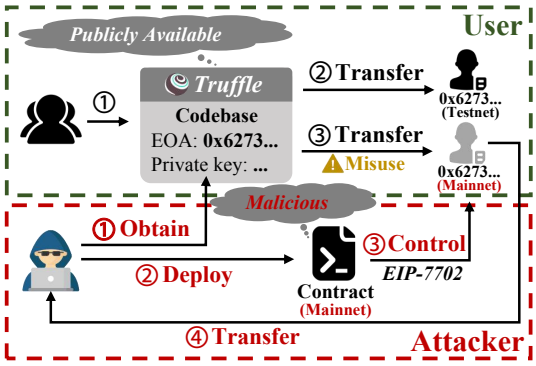

Figure 2: EOA Misuse and EIP-7702 Attack Process on Address `0x6273...Ef57` (Green line segment box: User-Induced EOA Misuse; Red line segment box: Attacker Exploitation via EIP-7702)

Example. As shown in the User part of the green line segment box in Figure 2, the address `0x6273...Ef57` is a well-known control-exposed address. This address originates from Truffle [56], where this address is included as one of the default accounts pre-generated for testing. Its private keys are exposed in the Truffle publicly available codebase [55]. Attackers can easily obtain the key by reading the source code or launching the Truffle development environment.

Users typically employ this default account for local testing. However, some mistakenly use it on the Ethereum mainnet[5], either due to confusion about network settings or the false assumption that the account is securely generated. Once funds are transferred to this address, any external attacker can use the exposed private key to take control and withdraw the

[2] https://etherscan.io/address/0xC532...4008
[3] https://etherscan.io/tx/0x2c2c...e214
[4] https://etherscan.io/tx/0x76d2...27eb
[5] https://etherscan.io/address/0x6273...Ef57

assets. On the Ethereum blockchain explorer Etherscan [21], it is tagged as "*Compromised*" and includes the warning: "*DO NOT SEND FUNDS TO THIS ACCOUNT! The seed phrase for this account is widely known, and any funds sent to this address will be immediately stolen*".

4 New Attacks Exploiting Address Misuse

Our measurement in Section 3 focuses on the phenomenon of address misuses. In practice, the most common cause of address misuses is user error, where, in the absence of any attacks, users mistakenly transfer their funds to incorrect addresses. While these *passive* address misuses have led to widespread and significant consequences, we also find that, in many cases, address misuses can be *actively* exploited by attackers for financial gain, thereby amplifying the security impact of such misuses. During our measurement study, we discover two new attack methods of actively exploiting Address Misuse to steal user funds, one corresponding to CA Misuse and the other to EOA Misuse. To the best of our knowledge, these attacks are not fully disclosed in prior research. Notably, both methods represent only special cases of Address Misuse, which involves an attacker maliciously leveraging mechanisms such as CA calculation or EIP-7702 to set address "traps" and attract users into interacting with them during address misuse. While our work focuses on measuring the impact of Address Misuse, the two new attacks also provide deeper insights and findings about security implications of address misuses.

4.1 CA Misuse Attack via CA Calculation

CA Misuse enables a new attack method that has not been disclosed before. Attackers can register a test version of the popular contract on the testnet and disseminate it, attracting users to use it. Then, some users may mistakenly use this address on the mainnet and experience CA Misuse, leading to a loss of real funds. Once a sufficient amount of funds accumulates at this address, the attacker can exploit the CA calculation mechanism to deploy a carefully crafted malicious contract at the same address on the mainnet, stealing all the misused funds.

We continue with the address example in Section 3.2. As shown in Figure 1, the UniswapV2Router02 contract on the testnet (`0xC532...4008`) is highlighted in the "Attacker" section of the red line segment box. After a notable amount of ETH accumulates at the address on the mainnet due to user CA Misuse, the deployer of the testnet `UniswapV2Router02` contract, i.e., the attacker, deploys a malicious contract to the same mainnet address on October 6, 2024 (mainnet block height: 20,902,977). The attacker achieves this by exploiting the deterministic CA derivation mechanism described in Section 2.2. As long as the deployer account and its nonce are the same, the contract address remains identical. We decompile the malicious contract bytecode, and its simplified core logic is shown in Figure 3. After successfully deploying the malicious contract, the attacker calls its `withdraw` function[6]. This transaction drains all the previously trapped funds from the address, totaling 3.78 ETH.

```
contract MaliciousWithdraw {
  address _withdraw;
  function withdraw() public payable {
    (v0,)=_withdraw.call().value(this.balance);
    require(bool(v0));}
  function withdrawETH(address to, uint256 amount)
       public payable {
    require(msg.sender == _withdraw,"only owner");
    (v0,)=to.call().value(amount).gas(msg.gas);} }
```

Figure 3: Decompiled malicious contract at the `0xC532...4008` address on Ethereum

It is worth noting that the official `UniswapV2Router02` address on Sepolia testnet provided by the official Uniswap documentation is `0xeE56...CfE3` [59]. This fact further confirms that the misused address `0xC532...4008` is likely deployed and propagated by an attacker. It also highlights the risk of using addresses from unofficial sources without careful verification against project documentation.

4.2 EOA Misuse Attack via EIP-7702

Recent upgrades on Ethereum and BNB Chain introduce the EIP-7702 proposal [11], which brings a new attack vector for EOA Misuse. Attackers can construct EIP-7702 transactions to delegate exposed EOAs to pre-deployed malicious contracts. Any funds transferred to the EOA subsequently trigger the delegated malicious contract, which automatically transfers them out.

The new attack vector introduced by EIP-7702 could make the EOA Misuse attacks more atomic and profitable. Normally, to conduct an EOA Misuse attack, the attacker must monitor on-chain transfer operations to addresses with exposed private keys. Once such incoming transfers are detected, the attacker needs to compete in gas wars, e.g., by using MEV bots, to withdraw the trapped funds before other potential attackers do. However, with the new attack vector introduced by EIP-7702, the transfer-in and transfer-out of funds occur within a single transaction, avoiding the need to wait for the user's deposit transaction to finalize before the attacker initiating a withdrawal. Additionally, it eliminates the need to compete in gas wars with other MEV bots, allowing the attackers to keep the saved gas fees as additional profit.

We continue with the EOA example in Section 3.3, i.e., the default account from the Truffle (`0x6273...Ef57`), as shown in the Attacker part of the red line segment box in Figure 2. In

[6] `https://etherscan.io/tx/0xbe40...7cae`

its recent transaction history[7], we observe this new EIP-7702 based attack. The attackers use its exposed private key to construct, sign, and submit an EIP-7702 transaction, delegating it to a malicious contract pre-deployed at another address. As a result, the EOA account becomes fully controlled by the malicious contract. The core logic of the contract after decompiling is shown in Figure 4. When users send funds, it automatically triggers the malicious contract's `receive()` function, transferring the funds to another address.

```
contract CrimeEnjoyor {
  address public destination;
  receive() external payable {
    require(destination != address(0), 'Not
        initialized');
    payable(destination).transfer(msg.value);} }
```

Figure 4: Malicious contract delegated by `0x6273...Ef57` address via Eip-7702 on Ethereum

5 Detecting Real-World Address Misuses

To detect real-world address misuses and evaluate their prevalence and security impact, we propose an approach for address misuse detection, which combines off-chain address data mining and on-chain transaction and contract analysis.

5.1 Overview

As shown in Figure 5, our workflow for detecting address misuse consists of four main stages. First, we begin by collecting addresses and private keys from mainstream developer platforms, such as GitHub and Stack Exchange. Then, the on-chain data fetching component leverages the Etherscan API to retrieve the complete transaction history and any existing bytecode of the collected addresses. Next, as the core stage of the workflow, the misuse analysis component examines the retrieved on-chain data. Specifically, it iterates through all transactions for each candidate address, matching them against predefined misuse patterns. Lightweight symbolic execution is also employed to determine whether the associated smart contract contains malicious logic. Finally, it records all detected misuse addresses, quantifies the total financial losses, and outputs the final detection results.

5.2 Data Collection

Our analysis focuses on platforms that may propagate addresses that can lead to misuse. We collect our candidate address and private key set from two primary sources: the code hosting platform GitHub [32] and the programming Q&A community Stack Exchange [31].

[7] https://etherscan.io/tx/0x808a...fa4d

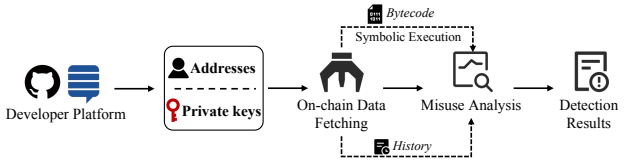

Figure 5: The Overall Workflow for Measuring Address Misuse Loss.

Public Code Repositories. We perform large-scale data mining on GitHub. Using blockchain-related keywords such as "ethereum" and "solidity" as tags, we search and download repositories created between January 2015 and May 2025 through the GitHub API [33]. To ensure broad coverage, we apply a snowball sampling strategy, iteratively discovering new repository tags to expand our search scope. All used tags are available in our open-source repository.

Online Q&A Communities. We utilize the public data dump from Stack Exchange. Specifically, we download the full archive provided by the Internet Archive as of April 2025 [6]. We select data from two sub-sites: Ethereum Stack Exchange [30] and Stack Overflow [46]. The former is dedicated to Ethereum-related programming questions, while the latter contains a wide range of technical programming content, including Ethereum topics. Since both sites are part of the Stack Exchange network, we collectively refer to their content as StackExchange posts.

We use regular expressions to extract potential addresses and private keys from the collected data to build our candidate set. The specific regular expressions and our validation logic are detailed in the Appendix A.

5.3 Data Analysis

To develop effective detection rules, we first search for and manually audit a set of real-world misused addresses to summarize their on-chain behavioral patterns. For CA Misuse, we examine the official addresses of the top 100 most active projects on Ethereum and BSC, as their large user bases increase the likelihood of user misuse. We analyze whether these addresses are reused across chains, for example, when a project address on Ethereum is mistakenly used on BSC, and summarize the characteristic transaction patterns of such misuse cases. For EOA Misuse, we get the top 100 GitHub repositories under the "Solidity" tag, extract all exposed private keys, and analyze the transaction histories of their corresponding addresses to summarize typical behavioral patterns. Specifically, the transaction patterns of CA Misuse addresses fall into two main categories: (1) function calls to addresses that have no deployed code, and (2) function calls to addresses that have code but before its deployment. The characteristics of EOA Misuse address are that it shows a transaction pair pattern of rapid fund sweeping caused by the mev bots in its

transaction history, and has recently been delegated to a malicious EIP-7702 contract. Based on these audited ground-truth cases, two authors collaboratively formulate rules that capture the behavioral patterns of misused addresses, which we then apply to large-scale detection in Section 5.3.1 and 5.3.2.

5.3.1 Identifying CA Misuse

Our methodology for detecting CA Misuse is detailed in Algorithm 1. The process begins by classifying each candidate address into one of two subtypes based on its on-chain state, and then applies specific detection rules.

Algorithm 1: Detecting CA Misuse

Input: $\mathbb{A}$, Set of candidate addresses
Input: $\mathbb{T}$, Map of transaction lists for each address
Output: R, List of CA Misuse instances: (addr, subtype, loss)

$R \leftarrow \emptyset$
Function `IsCAMisuseCall`$(tx, addr)$**:**
  return $tx.to = addr \wedge$ `hasFunctionCall`$(tx.input) \wedge$ $\neg$`isHumanReadable`$(tx.input) \wedge tx.gas < 30000$
foreach $addr \in \mathbb{A}$ **do**
  $T_{addr} \leftarrow \mathbb{T}[addr]$
  $t_{create} \leftarrow$ `getContractCreationTxTime`(T_{addr}) `// Check contract creation tx to label addr as NCD or PCD.`
  if t_{create} *is Null* **then**
    $subtype \leftarrow$ "NCD"
    `// Rule 1: Function Call to No-Code Address`
    $T_{risk} \leftarrow \{tx \in T_{addr} \mid$ `IsCAMisuseCall`$(tx, addr)\}$;
  else
    $subtype \leftarrow$ "PCD"
    `// Rule 2: Premature Call to a Code-Present Address`
    $T_{risk} \leftarrow \{tx \in T_{addr} \mid$ `IsCAMisuseCall`$(tx, addr) \wedge tx.time < t_{create}\}$
  $loss \leftarrow \sum_{tx \in T_{risk}} tx.value$
  if $loss > 0$ **then**
    $R \leftarrow R \cup \{(addr, subtype, loss)\}$
return R

We define the two subtypes as follows:

- **No-Code at Detection (NCD):** This indicates the address hosts no contract code at the time of our detection. It could be an EOA or a pre-computed address that has not yet deployed any contract.
- **Present Code at Detection (PCD):** This indicates that the address contains deployed contract code at the time of detection, and misuse occurs prior to deployment. The subsequently deployed contract may be benign or maliciously planted by attacker. We distinguish this subtype to enable further analysis of active address misuse attacks involving malicious contract deployment.

For addresses classified as *NCD*, we apply Rule 1 (Function Call to No-Code Address). A transaction is flagged as a misuse if it contains a function call directed to an address with no contract creation history. Specifically, the `hasFunctionCall` function validates this by determining if the first 4 bytes (8 hex characters) of the `input` data, which represent the function selector, have a corresponding function signature. Furthermore, we filter out cases where the `input` field is used for sending messages, i.e., Input Data Messages (IDM) [22]; the `isHumanReadable` function performs this check by determining if the hexadecimal `input` data can be decoded into a readable sequence of printable characters.

For addresses classified as *PCD*, we apply Rule 2 (Premature Call to a Code-Present Address). It flags transactions that attempt a function call on a CA before the contract creation timestamp. This scenario serves as a key precondition for the new attack based on misuse and is highlighted in our analysis.

Fund-Sweeping Contract Analysis. For the *PCD* subtype addresses where contracts are deployed, we perform further analysis to determine whether these contracts are part of malicious attacks, i.e., they are used to sweep the trapped fund from the address. The attack follows a three-step pattern: (1) Users mistakenly send funds to the address in premature calls. (2) The attacker deploys a malicious contract after funds accumulate. (3) The attacker invokes the contract to drain all accumulated assets.

To confirm this, we develop a lightweight symbolic execution engine to analyze the bytecode of deployed contracts without relying on the source code. Our tool first disassembles the bytecode, partitions it into basic blocks, and constructs a Control Flow Graph (CFG). During path exploration, we prune infeasible paths using a small timeout (i.e., 1 second) to maximize efficiency. To further optimize performance, our lightweight engine selectively solves constraints only for opcodes related to external calls, which is sufficient to capture the transfer semantics required to identify fund-sweeping logic. Our lightweight engine balances precision and performance by avoiding the semantic unrecognition problems of bytecode pattern matching methods, while achieving higher efficiency than fully symbolic execution. For instance, when an execution path reaches a `CALL` opcode, the engine retrieves the transfer amount from the maintained stack, and compares it with the contract's entire balance (`address(this).balance`) or symbolic input parameters to determine whether a full balance drain is occurring, thus marking the execution path as malicious. To handle proxy patterns, our tool resolves the implementation address upon encountering a `DELEGATECALL`. If the address is hardcoded, we fetch its bytecode for fund-sweeping analysis; if it is loaded from a storage slot, we first retrieve the value from that slot and then fetch the bytecode. The presence of a `SELFDESTRUCT` opcode is also considered a definitive indicator of a fund-sweeping design. If our analysis discovers any of these patterns, we classify the contract as malicious.

5.3.2 Identifying EOA Misuse

Our method for identifying EOA Misuse consists of two parts: a direct analysis of known private keys and a pattern-based analysis of on-chain transactions. This process is demonstrated in Algorithm 2.

We classify the results into two subtypes:

- **KEY (Exposed Private Key):** This subtype confirms

Algorithm 2: Detecting EOA Misuse

Input: $\mathbb{K}_{cand}$, Set of candidate private keys
Input: $\mathbb{A}_{cand}$, Set of candidate addresses
Input: $\mathbb{T}$, Map of transaction lists for each address
Output: R, List of EOA Misuse instances: (addr, subtype, loss)

$R \leftarrow \emptyset$;
// Part 1: Direct Analysis to Private Keys
foreach $key \in \mathbb{K}_{cand}$ **do**
  $addr \leftarrow$ deriveAddress(key) **if** $addr \in \mathbb{T}$ **then**
    $loss \leftarrow \sum\{tx.value \mid tx \in \mathbb{T}[addr] \wedge tx.to = addr\}$
    if $loss > 0$ **then**
      $R \leftarrow R \cup \{(addr, \text{"KEY"}, loss)\}$
// Part 2: Pattern-based Analysis to Addresses
foreach $addr \in \mathbb{A}_{cand}$ **do**
  $T_{in} \leftarrow \{tx \in \mathbb{T}[addr] \mid tx.to = addr\}$;
  $loss \leftarrow 0$
  foreach $in \in T_{in}$ **do**
    // Identify immediate fund-sweeping behavior
    $out \leftarrow$ findSweepTxPair($in, \mathbb{T}[addr]$)
    if $out \neq Null \wedge$ isEIP7702Malicious($addr$) **then**
      $loss \leftarrow loss + in.value$;
  if $loss > 0$ **then**
    $R \leftarrow R \cup \{(addr, \text{"ADDR"}, loss)\}$
return R;

that control is exposed because we have successfully extracted the private key corresponding to the address.

- **ADDR (Pattern-based Identified Address):** This subtype represents a highly suspected exposure case identified through heuristic rules based on on-chain activity.

The Direct Analysis corresponds to the *KEY* subtype. For each private key in our candidate set, we derive its public address and sum the total value of all its incoming transactions. This sum represents the confirmed financial loss.

The Pattern-based Analysis corresponds to the *ADDR* subtype and is applied to the address candidate set. This analysis is necessary because relying solely on directly discovered private keys is incomplete. Some keys might be removed from public sources after their initial exposure, or their exposure could be more subtle, e.g., derived from a public mnemonic phrase or wallet recovery flaws. Therefore, we apply the summarized heuristic rules to the address candidate set to identify suspicious activity. As detailed in Algorithm 2, we iterate through the time-sorted transaction list of each candidate address, first identifying characteristic patterns of automated fund sweeping via the `findSweepTxPair` function. This function searches for pairs of incoming transactions (*in*) and subsequent outgoing transactions (*out*) that satisfy a set of strict conditions. Such transaction pairs match the characteristic behavior of MEV bot attacks, where funds are immediately transferred out after being received. We also check additional conditions, such as the `gasPrice` of the transaction pair and the `balance` of the address. The specific logic of `findSweepTxPair` is provided in Appendix D. We next apply the `isEIP7702Malicious` function to check if the address is delegated to a malicious contract through the EIP-7702 mechanism. This novel exploitation method is widely adopted in automated attacks against addresses with exposed private keys. If an address satisfies all the above rules, we label it as an EOA Misuse case and calculate the fund loss.

EIP-7702-based Attack Analysis. As part of our detection algorithm, `isEIP7702Malicious` leverages lightweight symbolic execution to identify whether an EIP-7702 delegated contract exhibits malicious behavior. It first retrieves the code at the address. As described in Section 2.4, if the returned bytecode begins with the prefix `0xef0100`, it indicates that the account has delegated its control to a contract via EIP-7702 mechanisms. We then perform lightweight symbolic execution on the delegated contract. The analysis focuses on whether the fallback or receive function implements malicious logic to transfer the contract's entire balance or the incoming msg.value, enabling the contract to immediately transfer funds to an attacker-controlled address upon receipt.

6 Evaluation

We outline the following Research Questions (RQs) to evaluate the impact of Address Misuse and the effectiveness of our detection methods.
RQ1: Can our approach effectively detect Address Misuse?
RQ2: How prevalent and what impact does Address Misuse have in real-world on-chain transactions?
RQ3: Can attackers exploit Address Misuses to make profits?

6.1 Dataset Construction

To perform our evaluation, we first conduct a large-scale data collection from GitHub and Stack Exchange using the method in Section 5.2. We gather 63,004 GitHub repositories, from which we extract 10,312,572 unique addresses and 16,305,399 private keys after deduplication. From Stack Exchange posts, we collect 15,342 addresses and 75,057 private keys. Then, we conduct a large-scale on-chain analysis on Ethereum and BSC. In the GitHub dataset, we find that 5,018,957 addresses and 4,189 private-key-matched addresses have transaction records on Ethereum; 2,552,183 addresses and 12,660 private-key-matched addresses have transactions on BSC. In the Stack Exchange dataset, 6,044 addresses and 454 private keys are active on Ethereum, and 2,946 addresses and 668 private keys are active on BSC. The total scale of our data gathering and analysis reaches 53,416,740 addresses instances and $5.76 * 10^9$ transactions.

We applied the detection methods in Section 5.3 to these addresses to identify potential misuse instances. Our framework exhibits high scalability and efficiency, processing addresses in an average of 0.09 seconds, as 94.56% of them have fewer than 10,000 transactions. The results are summarized in Table 3, categorized by misuse type, blockchain, and data source. We further classify Address Misuses into subtypes as described in Section 5.3.1 and 5.3.2 to present the loss results. We use the token price at the time of writing (May 2025) as a reference, specifically $4,408 per ETH and $847 per BNB.

6.2 RQ1: Precision of our Approach

Before analyzing the specific impact of Address Misuse, we first perform manual sampling validation to answer RQ1, in order to assess the precision of Address Misuse instances identified by our automated analysis workflow. A high-precision detection result is fundamental to ensuring the reliability of our quantified financial losses. Since the *KEY* subtype under EOA Misuse involves actual leaked private keys, its results are 100% accurate. Therefore, our manual verification focuses on the results for CA Misuse and the *ADDR* subtype under EOA Misuse.

Evaluation Setup. Following standard practices from prior work [39,52], we set a 5% confidence interval and a 95% confidence level, performing random sampling stratified by misuse type and blockchain network. Two authors independently conduct a detailed manual analysis of each sampled address. To ensure the objectivity and reliability of our manual labeling, we adopt the Kappa coefficient to evaluate inter-rater reliability. The analysis process includes: (1) Tracing the original context of the address from GitHub or Stack Exchange to determine whether the address is explicitly linked to a specific blockchain. (2) Checking the tags on Etherscan or Bscscan that might indicate the address's identity or flag potential risks, e.g., "Compromised" or "Warning". (3) Reviewing the complete transaction history to evaluate whether the address behavior aligns with the corresponding misuse definition. (4) If the address hosts a contract, we decompile and analyze the bytecode to assess whether its functions are malicious. The analysis of code is further detailed in RQ3.

Table 1: Sampling and Evaluation Results

Misuse Type	Chain	Total	Sample Size	TP	FP	Precision (%)
CA Misuse	ETH	27,569	379	370	9	97.62%
	BSC	21,775	378	374	4	98.94%
EOA Misuse -ADDR	ETH	3,893	350	350	0	100.00%
	BSC	4,826	356	356	0	100.00%

Precision. During the labeling process, we classify results as True Positives (TP) or False Positives (FP) to evaluate detection performance. The two authors compare their results, achieving a high Kappa score of 0.868, and discuss any conflicting cases to reach a consensus. Additionally, we manually inspect and validate the top 10 addresses and transactions by loss amount for each misuse type on Ethereum and BSC to ensure that high-loss cases are genuine. Table 1 presents the results of manual validation. Our detection method achieves an overall precision of 99.11%. For the CA misuse type, the detection precision reaches 97.62% and 98.94% on ETH and BSC, respectively. For the *ADDR* subtype, it reached 100%.

False Positives. During the manual validation, we also identify a few FPs. These cases stem from transactions that users send to NCAs. Although the input data field in these transactions is not empty, it is not a valid function call but rather custom information or remarks used for specific purposes. While we filter such cases by checking if the input data is human-readable and whether the function selector corresponds to a known function signature, a few edge cases remain. For example, a user sends a transfer with `0xdeadbeaf` as a marker, which our detection misinterprets as a call to the function `CodeIsLawZ95677371()`, resulting in a false misuse classification.

In Appendix C, we further present two real-world cases for both CA Misuse and EOA Misuse, each resulting in extremely significant financial losses. These cases have not been fully disclosed before, highlighting the severity of Address Misuse and the effectiveness of our approach.

Table 2: Comparison Between Our Study and [66] on Leaked Private Keys Extracted from GitHub for Ethereum and BSC

Comparison Metric		Our Study	Study [66]
All Keys		16,305,399	8,650,000
ETH	Active Keys	3,765	1,370
	Transactions	432,902	425,000
	Value/Loss	103,402.5 ETH ($456M)	$7M
BSC	Active Keys	3,452	1,484
	Transactions	310,445	42,100
	Value/Loss	8,521.1 BNB ($7.2M)	$590k

Comparison with Existing Results. We further evaluate our approach by comparing our results with related work. Since no prior work reveals the CA Misuse phenomenon, our comparison focuses on the EOA Misuse. The work in [66] is similar to our analysis of the *KEY* subtype, examining leaked private keys from GitHub and their losses on Ethereum and BSC. The comparison results are shown in Table 2. Our findings uncover more exposed private keys and associated losses. This may be because our dataset is more up-to-date, covering more subsequently leaked keys and more transactions, which indicates that the problem is not being effectively mitigated. Furthermore, we extend the scope to include CA Misuse and the *ADDR* subtype under EOA Misuse, both of which are not covered in prior studies.

> **Answer to RQ1:** Our detection approach achieves an overall precision of 99.11%. It validates the effectiveness of our methods in identifying real-world Address Misuses and justifies the evaluation results in RQ2.

6.3 RQ2: Prevalence and Impact

To answer RQ2, we quantify the number of addresses and transactions affected by both types of Address Misuses, as

Table 3: Statistics on addresses, transactions, and losses related to Address Misuse on the Ethereum mainnet and BSC. The Total column reports deduplicated counts of addresses, transactions, and losses.

Type	Chain	Source	Subtype	Instances	Transactions	Loss	Total (dedup.)
CA Misuse	ETH	GitHub	PCD	582	40,100	3,499.85 ETH	27,569
			NCD	26,908	694,886	19,229.40 ETH	737,075
		Stack Exchange	PCD	31	12,945	2,706.62 ETH	22,738.41 ETH
			NCD	791	189,634	1,014.35 ETH	
	BSC	GitHub	PCD	576	45,229	1,167.09 BNB	21,775
			NCD	21,160	809,773	7,513.66 BNB	856,090
		Stack Exchange	PCD	40	35,638	719.16 BNB	8,681.41 BNB
			NCD	856	262,330	714.66 BNB	
EOA Misuse	ETH	GitHub	KEY	3,765	432,902	103,402.53 ETH	7,699
			ADDR	3,852	55,608	532.24 ETH	492,238
		Stack Exchange	KEY	375	165,832	46,390.73 ETH	104,244.53 ETH
			ADDR	56	35,152	1,279.06 ETH	
	BSC	GitHub	KEY	3,452	310,445	8,521.07 BNB	8,297
			ADDR	4,796	98,426	391.90 BNB	415,372
		Stack Exchange	KEY	556	91,957	467.01 BNB	9,045.29 BNB
			ADDR	40	12,778	48.34 BNB	

well as the corresponding financial losses, to assess the prevalence and security impact of real-world Address Misuses. The results are summarized in Table 3. The `Total` column shows the deduplicated number of misuse addresses, related transactions, and final losses for each misuse category on Ethereum and BSC.

Overall, we find substantial asset losses due to Address Misuses. We identify 65,340 high-risk addresses instances, with $2.5 * 10^6$ related misused transactions, resulting in a total loss of 126,982.94 ETH and 17,726.7 BNB, equivalent to over $5.75 * 10^8$ USD. The volume of affected addresses and transactions and the scale of losses confirm that this is not an isolated incident but a systemic security risk.

6.3.1 Impact Analysis of CA Misuse

The results concerning CA Misuse are presented in the upper part of Table 3. For clarity, each row in this section provides statistics for a specific combination of blockchain (Ethereum or BSC), data source (GitHub or Stack Exchange), and misuse subtype (*PCD* or *NCD*). For instance, the first row shows that on the Ethereum (ETH) chain, we identified 582 instances of the *PCD* subtype from GitHub, which were involved in 40,100 transactions and resulted in a loss of 3,499.85 ETH.

The data shows that CA Misuse is highly prevalent, with about 49,344 unique risky addresses instances and $1.6 * 10^6$ related transactions across both chains, resulting in a loss of 22,738.41 ETH on Ethereum and 8,681.41 BNB on BSC after deduplication. Among the subtypes, *NCD* addresses from GitHub dominate in both scale (26,908 in Ethereum and 21,160 in BSC) and financial impact (19,229.40 ETH and 7,513.66 BNB). This demonstrates that users are highly susceptible to misusing addresses propagated on GitHub, likely due to code reuse. In contrast, *PCD* addresses are fewer in number, but exhibit a higher average loss per instance (e.g., 582 Ethereum instances result in 3,499.85 ETH in losses). A plausible explanation is that these addresses are related to situations where attackers deploy contracts to drain after accumulating a significant amount of misused funds, which allows them to capture a larger value in a single event. While the number of instances and losses from Stack Exchange are smaller, they still represent a non-negligible risk vector.

> **Insight 1:** The results reveal the prevalence of CA misuse. Losses of 22,738.41 ETH and 8,681.41 BNB highlights the systemic error of users not verifying the target address and network, which often leads to mistakenly treating NCAs as CAs.

6.3.2 Impact Analysis of EOA Misuse

The lower part of Table 3 details the findings for EOA Misuse. As shown in this table, EOA misuse leads to even more severe losses than CA misuse, affecting about 15,996 addresses and $9.1 * 10^5$ related transactions, with total losses amounting to 104,244.53 ETH and 9,045.29 BNB after deduplication. The most alarming finding is the direct impact of *KEY* subtype from GitHub. This single misuse vector is responsible for large losses of 103,402.53 ETH and 8,521.07 BNB, accounting for over 95% of all EOA Misuse losses. This highlights a

Table 4: Sampling and Evaluation Results of Malicious Contracts Detection in the PCD Subtype of CA Misuse

Type	Chain	All	Found	Sample	TP	FP	TN	FN	Accuracy	Precision	Recall	F1-Score
CA Misuse -PCD	ETH	586	273	83	31	6	44	2	90.37%	83.78%	93.94%	88.57%
	BSC	576	196	83	24	3	51	5	90.36%	88.89%	82.76%	85.72%

critical security issue where users are widely interacting with these addresses that expose private keys, without realizing that they are insecure. In comparison, the identified *ADDR* subtype instances are smaller in scale, but still contribute to millions of USD in losses.

Insight 2: The findings reveal the prevalence of EOA Misuse. User interactions with exposed addresses are common, resulting in losses of about 104,244.53 ETH and 9,045.29 BNB. This reflects users' lack of awareness of address security.

Among the 10.3M addresses and 16.3M private keys collected from GitHub, only around 5M addresses on Ethereum and 2.6M on BSC show transaction activity. This indicates the presence of millions of dormant addresses. These may be used on other chains or testnets, or are simply inactive. Such addresses can be triggered into traps due to incorrect interactions by users or developers.

Insight 3: Despite detecting prevalent address misuses, over 85% of collected addresses have not yet attracted any misuse interactions, creating large-scale and potential risks if triggered by error interactions.

Answer to RQ2: Address Misuse is a prevalent and systemic security risk. It affects tens of thousands of addresses and leads to substantial financial losses, totaling over 126,982.94 ETH and 17,726.7 BNB on Ethereum and BSC, worth $574.8M.

6.4 RQ3: Real-world Attack Analysis

The *PCD* subtype under the CA Misuse category indicates that a contract is currently deployed at the misused address. These contracts include both normal ones and malicious ones deployed by attackers. Therefore, we first use symbolic execution described in Section 5.3.2 to distinguish the malicious contracts before analyzing their features. To ensure scalability and effectively mitigate the path explosion problem inherent in complex contracts, we explicitly configured our execution engine with strict constraints: a path exploration depth limit of 100, a timeout threshold of 500 seconds, and a loop unrolling bound of 10. Under these configurations, our tool achieved an average analysis time of 153.7 seconds per contract. It is important to note that even if an address now hosts a normal contract, it is still a CA Misuse instance as long as it meets the definition in Section 5.3.1, i.e., a function call occurs before the normal contract deployment. Our detection of malicious contracts does not affect the economic loss statistics in RQ2.

Accuracy in Detecting Malicious Contracts. To validate its accuracy, we perform a random sampling of contracts from all *PCD* subtype misuse addresses using a 10% confidence interval and a 95% confidence level. Two authors independently audit each sample by reviewing the decompiled code and on-chain interaction behavior to determine whether the contract is indeed used for fund theft. The results are shown in Table 4.

Our method for identifying malicious CA Misuse contracts achieves 90.37% accuracy, 83.78% precision, 93.94% recall and 88.57% F1-Score on Ethereum, and 90.36% accuracy, 88.89% precision, 82.76% recall, and 85.72% F1-Score on BSC , proving its reliability for subsequent analysis. The FPs are caused by contracts that are legitimate but implement a function to withdraw all funds. Our symbolic execution tools cannot detect such cases which require considering the overall semantic logic. FNs arise from contracts that extract funds using indirect methods, such as proportion-based withdrawals instead of using `this.balance` or explicit input amounts, which our current tool does not yet fully capture.

Characteristics of Malicious Contracts After confirming the set of malicious CA Misuse contracts, we classify their behavior characteristics to reveal the diverse strategies employed by attackers.

Table 5: Contract Pattern Distribution on ETH and BSC Chains

Chain	Contract Pattern	Num	Percentage
ETH	Direct Sweeper	236	86.45%
	Proxy	24	8.79%
	Self-Destruct	13	4.76%
BSC	Direct Sweeper	185	94.39%
	Proxy	8	4.08%
	Self-Destruct	3	1.53%

(1) Direct Sweeper. This is the most common pattern, accounting for 86.45% of Ethereum and 94.39% of BSC cases.

The attacker deploys a contract with simple withdrawal logic and calls it immediately after deployment to drain the funds.

(2) Proxy. 8.79% of attacks in Ethereum and 4.08% in BSC use a proxy pattern. The attacker deploys an open-source proxy contract as a disguise, which points to a non-open-source malicious logic contract. This helps evade simple user checks to gain trust and complicates attack tracing.

(3) Self-Destruct. 4.76% of malicious contracts in Ethereum and 1.53% in BSC use the `SELFDESTRUCT` function to transfer funds, attempting to erase on-chain evidence.

> **Insight 4:** 86.45% malicious CA Misuse contracts are simple direct sweepers, indicating that attackers only need minimal contract capabilities to successfully exploit them.

Official Recovery Contracts vs. Malicious Contracts. Notably, during our analysis, we find cases of official project teams deploying benign recovery contracts. Its purpose is to help users who mistakenly send assets to that address retrieve their funds. However, we also find cases of official teams deploying malicious extraction contracts that extract all misused funds without returning them to users. We discuss these cases in detail in Appendix C.

Code Clones in Malicious Smart Contracts Furthermore, our evaluation reveals widespread code cloning practices in malicious smart contracts, demonstrating how attackers systematically replicate malicious codes to exploit address misuses for profit. Specifically, we use the Similar Contract Search [23] feature provided by Etherscan and BscScan, using our confirmed malicious contracts as seeds for a search for exact matched contracts. The table 6 in Appendix B lists the top 5 malicious contracts with the largest number of exact code matches, i.e., clones. We manually inspect each contract and confirm that they are used for draining funds. For the top 5 malicious contract templates by clone count on each chain, we uncover 31,542 deployed instances (18,772 on Ethereum and 12,770 on BSC). This finding indicates that attackers reuse standardized malicious contract templates to systematically exploit large numbers of CA Misuse.

> **Insight 5:** Our findings uncover 31,542 malicious contracts cloned from just 10 unique templates, revealing that attackers create and extensively reuse malicious contracts to exploit CA Misuse at scale.

6.4.1 Contract Features of EOA Misuse Attacks based on EIP-7702

For EOA misuse attacks based on EIP-7702, we performed a cluster analysis of 7,824 maliciously delegated EOA addresses on Ethereum and 9,422 on BSC, along with the EIP-7702 malicious contracts they point to, observing a high level of centralization.

As shown in Table 7 in Appendix E, on Ethereum, we find 7,824 addresses with exposed control are delegated via EIP-7702 to only 49 unique malicious contracts. On BSC, 9,422 addresses are delegated to only 50 malicious contracts. The top 3 malicious contract addresses alone receive over 74.66% and 82.07% of all malicious delegations on Ethereum and BSC, respectively. This clearly demonstrates that a small number of attackers or groups are using standardized, reusable malicious contracts to systematically weaponize exposed private keys at scale through the EIP-7702 mechanism.

> **Insight 6:** The EOA Misuse attack based on EIP-7702 is highly scalable, with a small number of malicious contracts controlling most exposed EOAs.

The above analysis indicates that these attacks are not merely extreme cases of passive misuse; rather, they represent entirely new attack vectors that pose escalating threats. Specifically, our results show that CA Misuse attacks represent low-frequency yet high-impact threats, with 273 Ethereum (196 BSC) cases already causing total losses of 3,446.37 ETH (431.79 BNB). Additionally, although EIP-7702 exploited by EOA Misuse attacks was only introduced in Ethereum and BSC several months ago, EOA Misuse attacks have already emerged as growing risks, manifested in 7,824 Ethereum (9,422 BSC) cases.

> **Answer to RQ3:** We identify 469 unique malicious contracts that exploit CA Misuse, which have over 31,542 clones deployed on-chain. We also find 99 malicious contracts that control more than 17,246 EOAs via EIP-7702. This indicates that attackers systematically exploit Address Misuse with reusable and standardized malicious contracts to achieve efficient and large-scale attacks.

7 Discussion

7.1 Limitations

Data Completeness. We use GitHub and Stack Exchange as our sources for potentially risky addresses and exposed private keys. We acknowledge that the dissemination of such data also occurs on instant messaging platforms like Telegram or Discord. However, content on these platforms is typically ephemeral, private, and sparse, rendering it unsuitable for large-scale, reproducible measurement. Therefore, we prioritize public code repositories and Q&A platforms to ensure research reproducibility and ethical compliance. These sources provide sufficient real-world exposure for the first large-scale analysis. During data collection, some Github

repositories may be missed. To mitigate this, we employ a snowballing strategy to broaden search tags and maximize blockchain-related GitHub repository coverage.

Heuristic False Positives. Our heuristic rules for detecting the EOA Misuse *ADDR* subtype may misclassify certain special patterns. For example, some project wallets are managed by automated scripts whose behavior closely resembles MEV bots. To reduce such FPs, we introduce constraints such as requiring the outgoing transaction to drain nearly all funds and the address balance to remain extremely low. Our manual sampling validation confirms that FP rate remains low under the current rules, demonstrating its practicality and scalability.

Scope of Chains and Tokens. Our study currently quantifies the losses of native tokens (ETH and BNB) on Ethereum mainnet and BSC. Notably, Address Misuse is not unique to the EVM. Any account-based blockchain employing the same deterministic address derivation mechanism (e.g., between Solana and its testnet) is susceptible to this risk. Furthermore, both types of Address Misuse and corresponding novel attacks can also cause other standard tokens losses (e.g., ERC20 and ERC721). As the first study, we focus on the largest ecosystem (EVM) and the most widely used native tokens to establish a lower bound for this risk's severity. We plan to explore a broader range of chains and tokens in future work.

Loss Estimation. We quantify financial losses using token prices as of May 2025 ($4,408 for ETH and $847 for BNB). While our estimation may differ from historical value at the exact moment of the transaction due to market volatility, it provides a standardized metric to assess the current economic severity of these risks. Our manual validation confirms a detection precision of 99.11%, ensuring that the identified amount of ETH and BNB represent actively realized losses with minimal false positives. Nevertheless, there exists a gap between our observed loss and the total potential loss in the wild, suggesting that our figures represent a conservative lower bound. This underestimation stems primarily from the previously discussed limitations regarding Data Completeness and Scope of Chains and Tokens. Additionally, to ensure high precision, our heuristic rules prioritize specific, high-confidence patterns, which inevitably leads to false negatives by omitting subtler misuse cases or attacks that deviate from typical behaviors.

7.2 Mitigations

Our study reveals key blind spots in users' understanding of address security within the blockchain ecosystem. The findings provide warnings for end users, developers, and project teams. We propose the following recommendations to mitigate the problem of Address Misuse.

For Users. (1) Verify the network of an address. Users need to clearly recognize that the same address may represent entirely different entities across different networks. Always confirm that a contract address is valid for the intended chain. (2) Avoid using addresses from unofficial sources. Contract addresses should always be obtained from official project websites or documentation rather than unverified answers on social media or Q&A platforms. (3) Do not use test or example addresses in production. Accounts provided in development tools or sample code are for testing only and never use them with real assets in a mainnet environment.

For Developers. (1) Properly manage deployer addresses. Ensure the deployer account is used only for deploying contracts and does not initiate other transactions, thereby precisely controlling the deployer's nonce. Use the same deployer to deploy contracts across multiple major chains in a unified manner. This aims to maintain consistent contract addresses, reducing the risk of user misuse due to address discrepancies across chains. (2) Deploy Remedy Contracts. For established projects, teams can proactively deploy a protective remedy contract at the same address on other major public chains. This contract can provide refund mechanisms for mistaken interactions. (3) Proactively disclose and warn users about address information. Clearly indicate address–network mappings in documentation and website UI. Place explicit restrictions or warnings on the use of test accounts. (4) Prefer CREATE2 for contract deployment. This allows developers to redeploy updated contract versions at the same address, facilitating quick remediation and fund recovery after misuse. (5) Strengthen private key security management. When publishing open-source code or documentation, developers must ensure they remove all hardcoded private keys or mnemonic phrases. Sensitive information should be handled via environment variables or secure key management services, and team members should undergo continuous security training.

For Wallet Providers. Wallets should integrate real-time warning systems that query known misuse databases (such as our dataset) before signing transactions. If a user attempts to send transactions to an address that has no code on the current chain but exists as a contract on another chain, or interacts with an address whose private key exposed, the wallet should trigger a high-severity alert.

For Protocol. To mitigate cross-chain CA Misuse, future protocol upgrades (EIPs) should consider incorporating the chain ID into the address derivation formula for contract creation based on the CREATE opcode. This would ensure that a contract deployed by the same account with the same nonce results in unique, non-colliding addresses across different networks, effectively neutralizing cross-chain CA Misuse.

8 Related work

8.1 Private Key Leakage Detection

Several studies have investigated security risks caused by private key leakage in the blockchain ecosystem. Brengel et al. [8] analyze leaked Bitcoin keys on Pastebin. Other works [1, 2, 60] explore leakage by guessing or brute-forcing weak keys and wallet password. Zhou et al. [66] examine

Ethereum key leakage from major websites by matching private keys and quantify the resulting asset losses.

Differences. Our research is broader than prior work focused solely on private key leakage. We introduce and analyze another category of address risk, CA Misuse, which is not covered by existing literature. Furthermore, even within the scope of private key leakage which we term EOA Misuse, our work provides deeper insights: we analyze the typical transaction patterns of exposed addresses and the behavioral characteristics of attackers, enabling more effective detection of such risks. We also reveal a novel attack vector that weaponizes the EIP-7702, exploring a new frontier of this threat.

8.2 Maximal extractable value (MEV)

MEV remains a central issue in blockchain security. Several prior works [15, 18, 53] focus on front-running types of MEV activities. Others [7,49,64,65] investigate specific types of MEV, such as arbitrage using Automated Market Makers (AMMs) or flash loans. Qin et al. [48] focus on three MEV behaviors, including sandwich attacks, liquidations, and arbitrage. Li et al. [41] study DeFi MEV activities under the Flashbots bundle mechanism.

Differences. We focus on the fund-sweeping pattern of MEV that occurs within the specific context of EOA Misuse. It has not been investigated in previous MEV studies.

8.3 Address Poisoning Attacks and Scams

Guan et al. [34] and Tsuchiya et al. [57] study address poisoning attacks, which is another type of address interaction error. It exploit address truncation in wallet UIs and Ethereum block explorers to create deceptive addresses, polluting victim's transaction history and tricking them into sending funds to similar-looking addresses. In addition, numerous previous studies [5, 14, 36, 40, 62] investigate other scams targeting blockchain users. Scammers obtain users' private keys through phishing websites, impersonation of technical support staff, or social engineering, thereby gaining control of their assets. Honeypot attacks represent another sophisticated scam where malicious smart contracts with seemingly exploitable vulnerabilities trap victims' funds through hidden code mechanisms [12, 13, 54].

Differences. Our work is complementary yet distinct from these studies. Address poisoning and Address Misuse both stem from user negligence in address interaction, but they differ fundamentally. Address poisoning misleads users into interacting with a wrong address on the correct chain, while Address Misuse involves the correct address used in the wrong chain network context. The attack method for the latter is more subtle and complex. Honeypot attacks use complex contract logic to deceive users, but Address Misuse does not depend on the contract but the user's own mistakes. Furthermore, to the best of our knowledge, other scam studies do not cover scams related to CA Misuse.

9 Conclusions

Address Misuse are a widespread yet often overlooked attack surface in the Web3 ecosystem. This study systematically defines and analyzes two main types of Address Misuse, CA Misuse and EOA Misuse. We reveal their formation mechanisms, manifestations at the transaction level, and how attackers actively exploit user mistakes or inherent address risks to steal funds. We reveal two novel attack vectors corresponding to these two misuse types. Through large-scale data analysis using techniques such as heuristic rules, transaction pattern analysis, and lightweight symbolic execution, we quantify the financial losses caused by these misuse on Ethereum and BSC. Our findings show that their impact is both widespread and significant. This research not only provides methodologies and empirical data for identifying and mitigating Address Misuse but also offers specific and practical security recommendations for users and developers. Our goal is to raise the community's awareness of address security and contribute to building a safer blockchain ecosystem.

Acknowledgments

This work is supported by the Zhejiang Province "JianBingLingYan+X" Research and Development Plan (2025C02020), the National Natural Science Foundation of China (No. 62332004), "A New Technology Architecture for Web 3.0 Based on Blockchain" (2023YFB2704200), and the Open Research Fund of The State Key Laboratory of Blockchain and Data Security, Zhejiang University. Daoyuan Wu's role in this research is partially supported by Lingnan University's Direct Grant (ref. no.: DR26F6) and Faculty Research Grant (ref. no.: SDS25A13).

Ethical Considerations

In accordance with the USENIX Security ethics guidelines, we provide a structured discussion of the ethical implications of our study and the corresponding mitigation measures taken for different stakeholders.

Developers and Project Maintainers. Although blockchain addresses are generally anonymous, we are making a best-effort attempt to contact the developers of affected projects (e.g., via GitHub or Etherscan labels) to help them mitigate further potential harm.

Wallet Providers and Ecosystem Operators. We initiated a responsible disclosure process, sharing our findings, detection methods, and feasible misuse mitigations with wallet developers and exchanges. Our efforts enable them to deploy large-scale mitigations (e.g., wallet-level warnings) to protect users. Beyond disclosure, we are also exploring protocol-level

enhancements, e.g., adopting an Ethereum Improvement Proposal that incorporates the chain ID into the address derivation rule, to further mitigate cross-chain address confusion.

End Users. As the primary victims, users often misunderstand blockchain addresses and may reuse them across different chains, thereby risking fund loss. We plan to raise public awareness of this issue and release mitigation guidelines to help users avoid such errors.

Potential Attackers. To prevent potential misuse attacks by malicious attackers, we have omitted all sensitive details (e.g., private keys and exploit scripts) from the public release. Furthermore, we will request a publication embargo upon acceptance to provide developers with a critical window to deploy mitigations before our findings are fully disclosed.

Data Source Platforms. Our data collection was restricted to publicly available sources, including Stack Exchange, GitHub, and on-chain transactions accessible through public RPCs. We did not access any private, password-protected, or otherwise sensitive data, and our collection process caused no harm or disruption to any platform or user. We also plan to collaborate with these repository platforms to help them detect and warn about leaked blockchain credentials. Our study fully complies with institutional ethics requirements and applicable data privacy laws. No human subjects were involved.

Justice. Our work aims to enhance blockchain security and reliability for the broader community, rather than targeting specific individuals or projects. We uphold fairness, responsible disclosure, and respect for the public interest.

Open Science

Our dataset, analysis source code, and all results are publicly available at https://zenodo.org/records/17984167.

References

[1] Ethercombing: Finding secrets in popular places. https://www.ise.io/casestudies/ethercombing/, 2019.

[2] Stealing ethereum by guessing weak private keys, schneier on security. https://www.schneier.com/blog/archives/2019/04/stealing_ethere.html, 2019.

[3] 1inch. Difference between a wallet address, contract address, and a private key, 2024. https://help.1inch.io/en/articles/5711414-what-is-the-difference-between-a-wallet-address-contract-address-and-a-private-key.

[4] Binance Academy. Externally owned account, 2025. https://academy.binance.com/en/glossary/externally-owned-account-eoa.

[5] Bhupendra Acharya, Muhammad Saad, Antonio Emanuele Cinà, Lea Schönherr, Hoang Dai Nguyen, Adam Oest, Phani Vadrevu, and Thorsten Holz. Conning the crypto conman: End-to-end analysis of cryptocurrency-based technical support scams. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 17–35. IEEE, 2024.

[6] Internet Archive. Stack exchange data dump 2025-03-31, 2025. https://archive.org/details/stackexchange_20250331.

[7] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch Lafuente. Maximizing extractable value from automated market makers. In *International Conference on Financial Cryptography and Data Security*, pages 3–19. Springer, 2022.

[8] Michael Brengel and Christian Rossow. Identifying key leakage of bitcoin users. In *International symposium on research in attacks, intrusions, and defenses*, pages 623–643. Springer, 2018.

[9] BscScan. Bnb smart chain (bnb) blockchain explorer, 2025. https://bscscan.com/.

[10] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[11] Vitalik Buterin, Sam Wilson, Ansgar Dietrichs, and lightclient. Eip-7702: Set code for eoas, 2025. https://eips.ethereum.org/EIPS/eip-7702.

[12] Ramiro Camino, Christof Ferreira Torres, Mathis Baden, and Radu State. A data science approach for honeypot detection in ethereum. *arXiv preprint arXiv:1910.01449*, 2019.

[13] Weili Chen, Xiongfeng Guo, Zhiguang Chen, Zibin Zheng, Yutong Lu, and Yin Li. Honeypot contract risk warning on ethereum smart contracts. In *2020 IEEE International Conference on Joint Cloud Computing*, pages 1–8. IEEE, 2020.

[14] Zhuo Chen, Yufeng Hu, Bowen He, Dong Luo, Lei Wu, and Yajin Zhou. Dissecting payload-based transaction phishing on ethereum. *arXiv preprint arXiv:2409.02386*, 2024.

[15] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE symposium on security and privacy (SP)*, pages 910–927. IEEE, 2020.

[16] Ethereum docs. Ethereum accounts, 2025. https://ethereum.org/en/developers/docs/accounts/.

[17] Solidity 0.8.30 documentation. Contract abi specification, 2025. https://docs.soliditylang.org/en/v0.8.30/abi-spec.html#abi.

[18] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. In *International Conference on Financial Cryptography and Data Security*, pages 170–189. Springer, 2019.

[19] ethereum. go-ethereum, 2025. https://github.com/ethereum/go-ethereum/.

[20] Ethereum. Pectra, 2025. https://ethereum.org/en/roadmap/pectra/.

[21] Etherscan. Ethereum (eth) blockchain explorer, 2025. https://etherscan.io/.

[22] Etherscan. Idm, 2025. https://goto.etherscan.com/idm.

[23] Etherscan. Similar contract search, 2025. https://etherscan.io/find-similar-contracts/.

[24] Etherscan. Testnet sepolia (eth) blockchain explorer, 2025. https://sepolia.etherscan.io/.

[25] Ethereum Stack Exchange. Did i generate an existing ethereum address in parity?, 2017. https://ethereum.stackexchange.com/questions/16347/did-i-generate-an-existing-ethereum-address-in-parity/.

[26] Ethereum Stack Exchange. Hardhat local network keys generation, 2021. https://ethereum.stackexchange.com/questions/94886/hardhat-local-network-keys-generation.

[27] Ethereum Stack Exchange. Uniswap v2 router, factory on sepolia test network, 2023. https://ethereum.stackexchange.com/questions/150654/uniswap-v2-router-factory-on-sepolia-test-network.

[28] Ethereum Stack Exchange. What is the uniswapv2factory address on the sepolia network, 2023. https://ethereum.stackexchange.com/questions/151817/what-is-the-uniswapv2factory-address-on-the-sepolia-network/.

[29] Ethereum Stack Exchange. Uniswapv2 testnet support, 2024. https://ethereum.stackexchange.com/questions/161900/uniswapv2-testnet-support/.

[30] Stack Exchange. Ethereum stack exchange, 2025. https://ethereum.stackexchange.com/.

[31] Stack Exchange. Stack exchange, 2025. https://stackexchange.com/.

[32] Github. Github, 2025. https://github.com/.

[33] GitHub. Github rest api documentation, 2025. https://docs.github.com/en.

[34] Shixuan Guan and Kai Li. Characterizing ethereum address poisoning attack. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 986–1000, 2024.

[35] Hao Guo, Ehsan Meamari, and Chien-Chung Shen. Multi-authority attribute-based access control with smart contract. In *Proceedings of the 2019 international conference on blockchain technology*, pages 6–11, 2019.

[36] Bowen He, Yuan Chen, Zhuo Chen, Xiaohui Hu, Yufeng Hu, Lei Wu, Rui Chang, Haoyu Wang, and Yajin Zhou. Txphishscope: Towards detecting and understanding transaction-based phishing on ethereum. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 120–134, 2023.

[37] Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, et al. Kevm: A complete formal semantics of the ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 204–217. IEEE, 2018.

[38] Yoichi Hirai. Defining the ethereum virtual machine for interactive theorem provers. In *International conference on financial cryptography and data security*, pages 520–535. Springer, 2017.

[39] Molly Zhuangtong Huang, Rui Jiang, Tanusree Sharma, and Kanye Ye Wang. Exploring user perceptions of security auditing in the web3 ecosystem. *Cryptology ePrint Archive*, 2024.

[40] Xigao Li, Anurag Yepuri, and Nick Nikiforakis. Double and nothing: Understanding and detecting cryptocurrency giveaway scams. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2023.

[41] Zihao Li, Jianfeng Li, Zheyuan He, Xiapu Luo, Ting Wang, Xiaoze Ni, Wenwu Yang, Xi Chen, and Ting Chen. Demystifying defi mev activities in flashbots bundle. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 165–179, 2023.

[42] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254–269, 2016.

[43] NomicFoundation. hardhat, 2025. https://github.com/NomicFoundation/hardhat.

[44] NomicFoundation. hardhat accounts.ts, 2025. https://github.com/NomicFoundation/hardhat/blob/a1f64c64030496ad7062f56a752db7145f68dfff/packages/hardhat-core/test/internal/core/providers/accounts.ts#L40-L45.

[45] Openzeppelin. Deploying smart contracts using create2, 2025. https://docs.openzeppelin.com/cli/2.8/deploying-with-create2.

[46] Stack Overflow. Empowering the world to develop technology through collective knowledge, 2025. https://stackoverflow.co/.

[47] Parity. Parity technologies, 2025. https://www.parity.io/.

[48] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022.

[49] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. Attacking the defi ecosystem with flash loans for fun and profit. In *International conference on financial cryptography and data security*, pages 3–32. Springer, 2021.

[50] QuickNode. What is a nonce? management techniques for your ethereum transactions, 2025. https://www.quicknode.com/guides/ethereum-development/transactions/how-to-manage-nonces-with-ethereum-transactions.

[51] Reddit. Psa: if your account shows address "0x00a329c0648769a73afac7f9381e08fb43dbea72" do not deposit funds to it, 2017. https://www.reddit.com/r/ethereum/comments/6cixvy/psa_if_your_account_shows_address/.

[52] Tanusree Sharma, Zhixuan Zhou, Andrew Miller, and Yang Wang. A {Mixed-Methods} study of security practices of smart contract developers. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2545–2562, 2023.

[53] Christof Ferreira Torres, Ramiro Camino, et al. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1343–1359, 2021.

[54] Christof Ferreira Torres, Mathis Steichen, et al. The art of the scam: Demystifying honeypots in ethereum smart contracts. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1591–1607, 2019.

[55] trufflesuite. truffle/packages/hdwallet/test/hdwallet.test.ts, 2022. https://github.com/trufflesuite/truffle/blob/5b5312dbd5ffe2d5ec575e078ff09d98eb1b45c2/packages/hdwallet/test/hdwallet.test.ts.

[56] trufflesuite. truffle, 2023. https://github.com/trufflesuite/truffle.

[57] Taro Tsuchiya, Jin-Dong Dong, Kyle Soska, and Nicolas Christin. Blockchain address poisoning. *arXiv preprint arXiv:2501.16681*, 2025.

[58] Uniswap. Uniswap interface, 2025. https://app.uniswap.org/.

[59] Uniswap. Uniswap v2 deployment addresses, 2025. https://docs.uniswap.org/contracts/v2/reference/smart-contracts/v2-deployments.

[60] Marie Vasek, Joseph Bonneau, Ryan Castellucci, Cameron Keith, and Tyler Moore. The bitcoin brain drain: Examining the use and abuse of bitcoin brain wallets. In *International Conference on Financial Cryptography and Data Security*, pages 609–618. Springer, 2016.

[61] Alex Van de Sande Vitalik Buterin. Erc-55: Mixed-case checksum address encoding, 2016. https://eips.ethereum.org/EIPS/eip-55.

[62] Guoyi Ye, Geng Hong, Yuan Zhang, and Min Yang. Interface illusions: Uncovering the rise of visual scams in cryptocurrency wallets. In *Proceedings of the ACM Web Conference 2024*, pages 1585–1595, 2024.

[63] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International journal of web and grid services*, 14(4):352–375, 2018.

[64] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the just-in-time discovery of profit-generating transactions in defi protocols. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 919–936. IEEE, 2021.

[65] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 428–445. IEEE, 2021.

[66] Yuxuan Zhou, Jiaqi Chen, Yibo Wang, Yuzhe Tang, and Guofei Gu. Towards understanding crypto-asset risks on ethereum caused by key leakage on the internet. In *Companion Proceedings of the ACM Web Conference 2024*, pages 875–878, 2024.

A Extracting Private keys and Addresses

To extract addresses and private keys from GitHub repositories and StackExchange posts, we design and implement pattern-matching scripts based on regular expressions (regex). An Ethereum address starts with `0x` followed by 40 hexadecimal characters. The corresponding regex in Python is `(?:\b|^)(0x[a-fA-F0-9]{40})(?:\b|$)`. For private keys, the regex is `(?:\b|^)(?:0x|)([a-f0-9]64)(?:\b|$)`. If a matched address contains mixed-case letters, we verify its validity against the EIP-55 checksum [61]. If the address is in all lowercase, we conservatively include it in the candidate set. For extracted private keys, we compute the corresponding address and include it as well.

B Code Clones in Malicious Smart Contracts of CA Misuse

Table 6: Top 5 Identified Malicious Contracts

Chain	Identified Malicious CA	# Exact Matched CAs
ETH	0xac44...760b	17090
	0xf8df...bc42	1130
	0xb07e...4346	438
	0x50c7...b16a	67
	0x6448...5113	47
BSC	0xe53d...fbad	12259
	0xaea7...998b	177
	0xf9e0...d9d4	169
	0x04c1...1f85	92
	0x3d8f...1e83	73

C Case Studies

C.1 CA Misuse Cases

Case 1: `0x580D...375c`[8] has the largest total and single-transaction misuse loss. Upon investigation, this address corresponds to the Defi Protocol Aave LendingPool contract on the Ethereum Kovan testnet. However, on Ethereum mainnet, a user `0xd198...4FF0` mistakenly interacted with this address, which leads to a significant misuse loss. On September 7, 2020, i.e., at block 10815309, the user send a transaction attempting to call the deposit function, along with 2,580 ETH [9]. At that time, this address on mainnet is merely a PCA with no code deployed. Based on the technical principles described in Section 2.3, this transaction does not fail but is instead processed as a normal transfer, trapping the 2,580 ETH at the address. On August 3, 2021, i.e., at block 12950853, address `0x334E...6195` deploys an `InitializableAdminUpgradeabilityProxy` contract to this location. This proxy contract points to a logic contract `RescueImpl` at `0x3aE5...370e`. Figure 6 shows the main code of `RescueImpl`. The deployer then calls the rescue function[10], which transfers the funds to a hardcoded address `0x2517...7667`. We trace the subsequent flow of these funds and find that they are not returned to the original user `0xd198...4FF0` but are instead moved to cryptocurrency exchange addresses.

```
contract RescueImpl is VersionedInitializable {
  address public constant RESCUE_ADMIN = address(0
      x334E6291B73e340305f3FC5A65F4BCD3AA816195);
  address public constant FUNDS_RECEIVER = address
      (0x2517C251C8EDd3E6977051e6bb86Cc7876D07667)
      ;
  function rescue() public {
    require(msg.sender == RESCUE_ADMIN, '
        INVALID_CALLER');
    (bool success, ) = FUNDS_RECEIVER.call.value(
        address(this).balance).gas(50000)('');
    require(success, 'FAILED_TRANSFER');}}
```

Figure 6: Contract Code of `0x3aE5...370e` on Ethereum

Case 2: `0x10ED...024E`[11] is the official PancakeSwap Router contract on the BSC. However, this address is frequently misused on the Ethereum mainnet. It is labeled as "Pancakeswap: Router" on Etherscan, and the security tool MetaSuites marks it as "Fake Phishing". Both of these labels are incorrect, the address is actually an Address Misuse address. Starting on April 24, 2021, i.e., block 12300692, a large number of users cause misuse by calling functions like `swapExactTokensForTokens` and `swapExactETHForTokens`. It is not until September 10, 2024, that a Recovery contract is deployed to the address and the funds are transferred out [12]. Before this recovery, 6,128 misuse transactions occur, trapping a total of 94.99 ETH. The Recovery contract is shown in Figure 7. It can recover all types of tokens. We trace the subsequent fund flow and find that the PancakeSwap project team returns all the lost funds to the users. Furthermore, because the deployment of the Recovery contract, all future misuse transactions will revert, preventing any further losses for users.

[8] https://etherscan.io/address/0x580D...375c

[9] https://etherscan.io/tx/0xf58e...37de

[10] https://etherscan.io/tx/0x4f52...c599

[11] https://etherscan.io/address/0x10ED...024E

[12] https://etherscan.io/tx/0x14ca...ea74

```
contract Recovery is Ownable2Step {
  function recoverEth() external onlyOwner{
    (bool success, ) = payable(msg.sender).call{
        value: address(this).balance}("");}
  function recoverERC20(IERC20[] calldata tokens)
      external onlyOwner{...}
  function recoverERC721(IERC721[] calldata tokens
      , uint256[][] calldata tokenids) external
      onlyOwner{...}
  function recoverERC1155(IERC1155[] memory tokens
      , uint256[][] memory ids, uint256[][] memory
       values) external onlyOwner{...}}
```

Figure 7: Contract Code of `0x10ED...024E` on Ethereum

C.2 EOA Misuse Cases

Case 1: `0x00a3...EA72`[13] is generated by previous versions of Parity wallet [47] when an blank recovery phrase provided. As a result, anyone can easily gain control of this address [25, 51], including malicious attackers. The address has an accumulated total of 30,940 transactions on Ethereum, with losses of 5,224.92 ETH. A notable case occurred at block 4,230,203, where a victim transfers 500.2 ETH to the address [14]. Just two blocks later at block 4230205, an attacker spends 250.1 ETH in gas fees which worth $862,742.61 to extract the remaining 250.1 ETH balance [15]. Such an enormous gas fee is clearly not from a normal user-initiated transaction. We speculate that the attacker's MEV bot is designed to use half of the incoming funds as gas and transfer the other half, which guarantee inclusion in the block. This case also supports our design choice of setting block interval to 2 in Section 5.3.2, as such high-value MEV competition leads miners to wait for higher gas profits. Notably, Etherscan has no warning label for this address, leaving users without any alert. It has now been delegated to a malicious contract `0xB678...AA02` named `AdvancedCrimeEnjoyor` through the EIP-7702 mechanism. As shown in its code in Figure 8, the receive and fallback function ensure that any incoming funds are automatically forwarded to the address `0xA820...921D`.

```
contract AdvancedCrimeEnjoyor {
  receive() external payable {
    0xA8204582fe8f915b7f423b1450B654FC98fF921D.
        call{value: msg.value}(new bytes(0));}
  fallback() external payable {
    0xA8204582fe8f915b7f423b1450B654FC98fF921D.
        call{value: msg.value}(new bytes(0));}
  function transfer(address token, uint256 amount)
      public {}
  function transferFrom(address token, address
      sender, uint256 amount) public {}
  function transferEth(uint256 amount) public {}}
```

Figure 8: Contract Code of `0xB678...AA02` on Ethereum

[13] https://etherscan.io/address/0x00a3...EA72
[14] https://etherscan.io/tx/0x84c7...e7ff
[15] https://etherscan.io/tx/0xe790...051f

Case 2: `0xf39F...2266`[16] is one of the default accounts generated locally by the popular smart contract development tool Hardhat [43]. Similar to the Truffle, anyone can obtain the private key for this default account simply by launching the Hardhat environment [26] or viewing the code [44]. The difference is that this address does not yet have any warning labels on Etherscan. It has 2,265 transactions on Ethereum. Its largest loss occurs in transactions `0xbff8...645c` [17] and `0x3512...198d` [18], where 289 ETH are transferred in and 284.2 ETH are extracted with a high gas fee of 4.8 ETH which worth $16,618.82. This address is currently also delegated via EIP-7702 to a malicious contract at `0x0E04...4647`. The decompiled logic is shown in Figure 9.

```
contract MaliciousSweep{
  fallback() {
    v6=0xe6c9adf066484ee303deacdd0493bc8da1371c8b.
        call().value(msg.value).gas(msg.gas);
    require(v6);}}
```

Figure 9: Decompiled Code of `0x0E04...4647` on Ethereum

[16] https://etherscan.io/address/0xf39F...2266
[17] https://etherscan.io/tx/0xbff8...645c
[18] https://etherscan.io/tx/0x3512...198d

D Function findSweepTxPair

Process of findSweepTxPair. As shown in Algorithm 3, `findSweepTxPair` takes an incoming transaction *in*, and a list of subsequent transactions $subsequentTxs$ as input. Its purpose is to find a corresponding outgoing transaction *out*, which satisfies a set of conditions and exhibits the characteristics of fund sweeping. The specific parameters settings of these conditions are discussed later.

Algorithm 3: findSweepTxPair

Input: *in*, Incoming transaction
Input: T, Transactions list
Output: *out*, The matching outgoing transaction or **Null**

foreach $out \in T$ **do**
  $isOppositeFlow \leftarrow (out.from == in.to) \wedge (out.to \neq in.to)$
  $isInTimeWindow \leftarrow (out.block - in.block \in [0,2])$
  $isMevGasWar \leftarrow (out.gasPrice > 1.61 \times in.gasPrice)$
  $isDrainingFund \leftarrow (out.value + out.gasFee > 0.95 \times in.value)$
  $isDustLeft \leftarrow (\texttt{getBalance}(out.from) < 0.0001)$
  if $isOppositeFlow \wedge isInTimeWindow \wedge isMevGasWar \wedge isDrainingFund \wedge isDustLeft$ **then**
    return tx;
return Null;

The conditions checked by `findSweepTxPair` are as follows: (1) The fund-sweeping out occurs within a short time window after the in. Based on our observations, we conservatively set the time window to within 2 blocks to avoid missing potential cases (2) The address's balance remains consistently near zero due to continuous monitoring and draining. (3) Due to the MEV bots gas war for the transfer-out opportunity, the

gas price of out is significantly higher than that of in. The 1.61x threshold is derived from our observations and the iterative bidding model proposed by Qin et al. [1], representing the price increase after five rounds of bidding under Geth's default 10% price bump rule ($1.1^5 = 1.61$). (4) The transferred value plus gas in out is nearly equal to the value received from in, leaving only dust ETH/BNB. This means that the mev bot is trying its best to extract all the value.

Parameters Settings for Rule-based Detection. In the process of the `findSweepTxPair` above, a key parameter is the block interval between fund incoming and outgoing transactions, which we set to a maximum of 2 blocks. This setting is based on our observation of ground truth attack data. If the interval is limited to the same block or just 1 block, some typical large-value theft cases would be missed. For example, the 500.2 ETH loss from address `0x00a3...ea72` in the Case Studies C.2 occurs with an interval of 2 blocks. We hypothesize that in such large-value attacks, on-chain bidding among attackers is particularly intense. Blockchain miners or validators may wait longer for the bidding to maximize the gas fee and consequently, their own profit. Therefore, setting the upper limit at 2 blocks is an empirical choice that balances the FP and FN rates. Furthermore, we set the threshold for the gas price ratio between the outgoing and incoming transactions to 1.61. This is based on the iterative bidding model proposed in [48] and the default behavior of the Ethereum Geth client. Under Geth's [19] default settings, when a new transaction attempts to replace an existing one from the same address and with the same nonce in the Mempool, its gas price must be at least 10% higher. According to [48], most MEV bidding wars conclude within five rounds. As a conservative strategy, we set the threshold to $1.1^5 = 1.61$. Additionally, the Priority Gas Auctions (PGA) game-theory model in [15] suggests that mev bots eventually tend toward cooperative equilibrium, with bids converging around a 12.5% increment, which is the minimum markup required by the Parity client [47]. It is worth noting that this gas bidding heuristic is more effective for Ethereum's PoW era.

However, our detection parameters still maintain sufficient effectiveness under the Proposer-Builder Separation (PBS) architecture. Although MEV searchers can submit bundles and bribe builders to prioritize their transactions that sweep funds from exposed EOA addresses, these bribes can be trivially converted into increased gas costs. Consequently, competition through builder bribes converts into competition over gas prices. Furthermore, the race-to-empty nature of exposed keys compels MEV searchers to execute transactions in the earliest possible block, making delayed extraction (>2 blocks) rare. To further validate our method's post-PBS robustness, we analyze post-PBS activities of 100 EOA Misuse addresses of the *KEY* subtype, whose private keys are explicitly exposed, serving as ground truth. We find that 85.71% of withdrawal transactions still meet these detection parameters.

E EIP-7702 Malicious Contract Delegation of EOA Misuse

Table 7: Top delegated malicious contract addresses.

Chain	Delegated Malicious Contract Address	Delegation Count	Percentage (%)
ETH	`0x8938...e704`	2407	30.76
	`0xa03f...60ab`	2004	25.61
	`0x08f5...30f9`	1431	18.29
	`0xb678...aa02`	780	9.97
	`0x68ae...7351`	339	4.33
	`0x0e04...4647`	320	4.09
	`0x6b78...e933`	221	2.82
	`0x710f...8b6e`	164	2.10
	`0x8904...a629`	31	0.40
	`0x7f7f...44c8`	24	0.31
BSC	`0x15c4...bab3`	3154	33.47
	`0x1c36...763c`	2315	24.57
	`0xc99f...b54a`	2264	24.03
	`0x92fe...87af`	634	6.73
	`0x0610...0f19`	414	4.39
	`0x1107...b4d6`	169	1.79
	`0x8904...a629`	97	1.03
	`0x289c...8ae9`	67	0.71
	`0x3ae1...2d10`	57	0.60
	`0xccd8...fbcc`	48	0.51

F NOTATION SUMMARY

Table 8: Abbreviation Summary

Abbreviation	Description
EOA	Externally Owned Account
CA	Contract Account
NCA	Non-Contract Address
PCA	Potential Contract Address
NCD	No-Code at Detection (A subtype of CA Misuse)
PCD	Present Code at Detection (A subtype of CA Misuse)
KEY	Exposed Private Key (A subtype of EOA Misuse)
ADDR	Pattern-based Identified Address (A subtype of EOA Misuse)
BSC	BNB Smart Chain
EIP	Ethereum Improvement Proposal
EVM	Ethereum Virtual Machine
ABI	Application Binary Interface
IDM	Input Data Messages
PGA	Priority Gas Auctions
MEV	Maximal extractable value
AMM	Automated Market Makers