The Blockchain Warfare: Investigating the Ecosystem of Sniper Bots on Ethereum and BNB Smart Chain

FEDERICO CERNERA, Sapienza University of Rome, Italy
MASSIMO LA MORGIA, Sapienza University of Rome, Italy
ALESSANDRO MEI, Sapienza University of Rome, Italy
ALBERTO MARIA MONGARDINI, Sapienza University of Rome, Italy
FRANCESCO SASSI, Sapienza University of Rome, Italy

In the world of cryptocurrencies, the public listing of a new token often generates significant hype. In many cases, the price of the token skyrockets in a few seconds, and timing is crucial to determine the success or failure of an investment opportunity. In this work, we present an in-depth analysis of sniper bots, automated tools designed to buy tokens as soon as they are listed on the market. We leverage GitHub open-source repositories of sniper bots to analyze their features and how they are implemented. Then, we build a dataset of Ethereum and BNB Smart Chain (BSC) liquidity pools to identify operations performed using sniper bots. Our findings reveal 352,413 sniping operations on Ethereum and 1,716,917 on BSC for a total turnaround of $155,630,184 and $137,548,859, respectively. We find that Ethereum operations have a higher success rate but require a larger investment. Finally, we analyze possible countermeasures and mechanisms used in token smart contracts that can reduce the negative impact of sniper bots.

CCS Concepts: • **General and reference** → **Empirical studies**; **Measurement**; • **Information systems** → **Information extraction**; • **Applied computing** → **Economics**.

Additional Key Words and Phrases: Sniper bot, Ethereum, BNB Smart Chain, Cryptocurrencies, Smart Contract, Market manipulation.

ACM Reference Format:
Federico Cernera, Massimo La Morgia, Alessandro Mei, Alberto Maria Mongardini, and Francesco Sassi. 2025. The Blockchain Warfare: Investigating the Ecosystem of Sniper Bots on Ethereum and BNB Smart Chain. 1, 1 (January 2025), 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The cryptocurrency market is very well known for its high volatility [34], with cycles where the value of tokens can skyrocket and plummet [29]. This behavior is very common among tokens with small market capitalization, especially newly listed ones. The rapid increase in value often triggers FOMO [4, 35] (Fear Of Missing Out) among investors, who often purchase tokens based on hype rather than their intrinsic value. With the emergence of DeFi [65](Decentralized Finance) and specifically Automated Market Makers (AMMs) [63]—platforms where smart contracts power trading—every blockchain user can list and make their tokens tradable. As a result, hundreds of tokens are listed on AMMs

Authors' Contact Information: Federico Cernera, cernera@di.uniroma1.it, Sapienza University of Rome, Rome, Italy; Massimo La Morgia, lamorgia@di.uniroma1.it, Sapienza University of Rome, Rome, Italy; Alessandro Mei, mei@di.uniroma1.it, Sapienza University of Rome, Rome, Italy; Alberto Maria Mongardini, mongardini@di.uniroma1.it, Sapienza University of Rome, Rome, Italy; Francesco Sassi, sassi@di.uniroma1.it, Sapienza University of Rome, Rome, Italy.

Manuscript submitted to ACM

daily [62], and finding the next successful token can be challenging. This has created the ideal environment for the emergence of sniper bots—automated systems designed to buy tokens quickly as soon as they are listed on an AMM platform. In this work, we leverage open-source implementations of sniper bots to gain insight into their features. We find that sniper bot implementations are more sophisticated than we might expect. Indeed, some offer features such as protection against fraudulent liquidity pools (e.g., honeypots [59] and rug pulls [14]) and anti-bot mechanisms commonly implemented in token smart contracts. Then, we build the liquidity pools dataset, consisting of Ethereum and BSC liquidity pools and their activities. Inspired by what we learned analyzing the implementation of the sniper bots, we devised a straightforward approach to detect addresses that take advantage of sniper bots. The results of our analysis show that sniper bots phenomenon is more widespread on BSC than on Ethereum. However, after analyzing the operations conducted by these addresses, we surprisingly find that Ethereum operations are more likely to be closed with a profit despite requiring a larger investment. Finally, we leverage Etherscan and BscScan, two popular explorers for Ethereum and BSC, respectively, to retrieve the source code of the tokens' smart contracts in the liquidity pools dataset. We search among the retrieved smart contracts implementation for anti-bot mechanisms that can limit the action of the sniper bots. Our analysis reveals that token developers counter sniper bot activities by implementing one or more mechanisms to hinder their actions. Our main contributions are:

- **Analysis of sniper bots**: To the best of our knowledge, we are the first to conduct an in-depth analysis of sniper bots and their implementation. We explore each phase of a sniping operation, from the target liquidity pool's choice to the token's sale. For each phase, we detail the techniques implemented by the most popular open-source sniper bots on GitHub.
- **The impact of sniper bots**: We propose an identification methodology for buy and sell operations performed by sniper bots. We find 27,691 sniper bots on Ethereum and 44,471 sniper bots on BSC. Analyzing their operations, we note that sniper bot users behave differently according to the blockchain they operate. We analyze their operations to estimate their success rate and profits. We quantify their impact on the ecosystem of liquidity pools, finding that they move a volume of $155,630,184.2 (99,036.3 ETH) on Ethereum and $137,548,859.7 and (335,696.2 BNB) on BNB Smart Chain. Moreover, their actions are responsible for driving up the price of new tokens as soon as they are listed, often by more than 70%, creating an unfair market for human investors.
- **Smart contract analysis**: We describe the most popular mechanisms to counter bots used by smart contract developers. Then, we quantify the adoption of anti-bot mechanisms by tokens, leveraging a dataset of almost one million smart contracts. We observe that 34.4% token smart contracts on Ethereum and 36.2% on BSC implement at least one mechanism to hinder the action of bots.

2 Background

2.1 Ethereum

Launched in 2015, Ethereum [12] is a blockchain network that utilizes Ether (ETH) as its native cryptocurrency. ETH is the second-highest market capitalization among cryptocurrencies, exceeding $400 billion[1]. Initially, Ethereum relied on Proof-of-Work (PoW) for consensus. Under PoW, miners compete to solve complex puzzles, with the winner earning the right to add the following block to the blockchain. However, PoW has limitations in scalability and energy efficiency. Solving these puzzles requires significant computational power, leading to high energy consumption. To address these concerns, Ethereum transitioned to a Proof-of-Stake (PoS) system in September 2022. With PoS, instead of raw

[1]Data retrieved from CoinMarketCap [17] in 2024-03-26

computing power, the nodes that add the following block to the chain are chosen based on the amount of ETH they have staked. This shift reduces energy consumption and paves the way for better scalability.
Ethereum has two types of accounts: The Externally Owned Account (EOA) and the Contract Account. An Externally Owned Account (EOA) is defined by its key pair, composed of a public key serving as the account's identifier and a private key enabling the user to authorize transactions through digital signatures. Instead, a contract account serves as the on-chain representation of a deployed smart contract, acting as its unique identifier and point of interaction within the blockchain ecosystem. This account is automatically generated upon deploying the smart contract on the chain. Externally Owned Accounts (EOAs) and contract accounts utilize a standardized identifier of a 42-character hexadecimal string, distinguished by the "0x" prefix.

Within the Ethereum network, accounts can update the state of the Ethereum network by submitting transactions. Transactions are the fundamental mechanism for transferring assets, deploying smart contracts, and invoking smart contract functions. Each transaction specifies a sender (only EOAs can initiate a transaction and, therefore, be senders), a recipient, a value (representing the amount of the transferred asset), and a fee. The sender determines the transaction fee. The fee attached to a transaction directly influences its priority within the Ethereum network. Miners, the users responsible for validating and including transactions in the blocks, tend to prioritize transactions with higher fees to get higher rewards for their computational efforts. Consequently, transactions with higher fees are more likely to be included in the next block, leading to faster confirmation times. Furthermore, the fee reflects the computational resources required to process the transaction. When invoking smart contract functions, the complexity of the function directly impacts the fee. Senders must provide sufficient fees to cover these computational expenses.

One of the key features of Ethereum is the ability to run custom machine code via smart contracts. Specifically, smart contracts are programs written in a high-level programming language such as Solidity [20] or Vyper [55] and stored as bytecode on the blockchain. Smart contracts enable the creation of different types of assets, such as fungible and non-fungible tokens (NFTs) [32]. Tokens are assets created by blockchain users. The dynamics behind their trading mechanisms are defined via smart contracts. A smart contract is deployed on the blockchain by sending a transaction to the zero address; a unique address consisting of all zeros. The transaction must contain the bytecode of the smart contract. The ERC-20 [26] standard is crucial in defining a common interface for fungible tokens within the Ethereum ecosystem. This standard outlines the core methods and events that token contracts must implement. The methods provide basic functionalities such as transferring tokens from one account to another or obtaining an account's current token balance. On the other hand, the Events are a mechanism to ease the tracking of the internal state of smart contracts. Indeed, each time the state of the smart contract changes, it can trigger an Event that is written in the Event Log. Ethereum states are changed every time a block is added to the chain. The entity in charge of computing state changes is the Ethereum Virtual Machine (EVM) [61].

The EVM is used by numerous blockchains other than Ethereum, such as the BNB Smart Chain [5]. The EVM-compatible chains inherit key aspects of Ethereum's architecture, including the same format for EOA and smart contract addresses, the same transaction structures and fee mechanisms for processing transactions, and the use of the EVM to execute smart contracts.

2.1.1 Mempool. When a user initiates a transaction on the Ethereum network, it is first propagated to their directly connected nodes, known as neighboring nodes. These nodes perform preliminary verification checks, such as validating the transaction signature. Upon successful verification, the transaction is added to the node's local mempool, a temporary storage area for pending transactions. Subsequently, the node broadcasts the transaction to its neighbors, disseminating

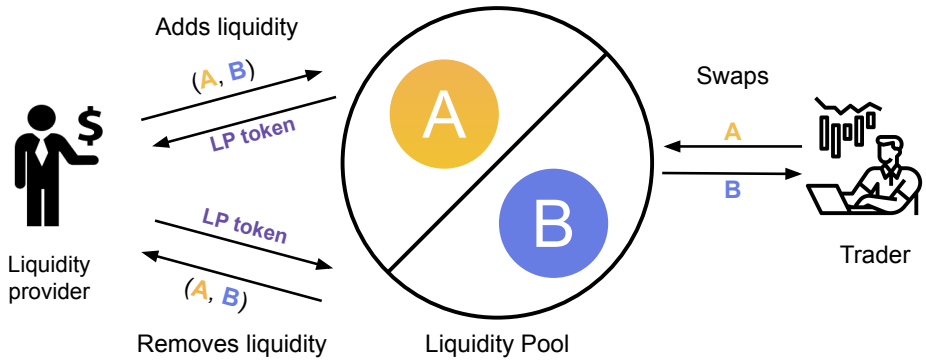

Fig. 1. An example of a liquidity pool and its main operations.

it throughout the network. It is important to note that the mempool is publicly accessible, allowing network observers to monitor unconfirmed transactions. Validators add new blocks to the Ethereum blockchain. They select transactions from their mempool to include in their proposed next block. While validators have discretion in choosing transactions and ordering within a block, economic incentives influence their decisions. Each transaction specifies a *gas limit* and a *gas price*, determining the transaction fee. Gas refers to the unit of computational effort required to execute the transaction, while the gas limit sets a maximum threshold for the fee the user is willing to pay. The gas price, denoted in Ether, determines the amount paid per gas unit. Miners, motivated by profit maximization, should prioritize transactions with higher gas prices.

2.2 Binance Smart Chain

The Binance Chain [5] was established in 2019 by Binance [7], one of the largest cryptocurrency exchanges in the world. The BNB Smart Chain (BSC) was later introduced as a parallel to the Binance Chain to support smart contracts. BSC uses the Proof of Stake and Authority (PoSA) [6] consensus, and its native coin, the BNB, is the third-largest cryptocurrency by market capitalization, with a value of over 86 billion USD[2]. The BSC is EVM-compatible (i.e., it leverages the same EVM of Ethereum). Thus, it can run Ethereum's smart contracts and has the same addresses and state management conventions. Note that BSC tokens utilize the BEP-20 standard, similar to Ethereum's ERC-20, but designed for the BNB Smart Chain. Because of this compatibility, the BSC is proposed as a faster and less expensive alternative to Ethereum.

2.3 DEX, AMM, and Uniswap

Decentralized exchanges (DEXs) are cryptocurrency exchanges that enable the trade of cryptocurrencies without the need for an intermediary. The most popular DEXs leverage the Automated Market Maker model (AMM) [64]. In this model, trade matching is carried out through liquidity pools, and the price of assets is determined by a mathematical formula. Uniswap [2] is one of the first decentralized applications (dApp) to use the AMM model successfully, with over 5.33 billion USD locked into smart contracts[3].

[2]Data retrieved from CoinMarketCap [17] in 2024-03-26
[3]Data retrieved from DefiLlama [36], a popular DeFi statistics aggregator, in 2024-03-26

A *liquidity pool* is a smart contract that contains a pair of ERC-20 tokens (A, B) that users can swap. Fig. 1 shows, at a high level, how liquidity pools work in Uniswap. Users who want to invest in the liquidity pool provide tokens (A, B) to the smart contract, becoming *liquidity providers*. Liquidity pools, to keep track of the share of the liquidity owned by each investor, use an ERC-20 token called LP-token. When a liquidity provider adds liquidity to the liquidity pool, the smart contract mints LP-tokens and transfers them to the liquidity provider. Conversely, a liquidity provider that wants to remove its liquidity can transfer the LP-tokens to the liquidity pool smart contracts. The smart contract burns the LP-tokens and returns the tokens (A, B) back to the investor.

Any user can interact with the liquidity pool to swap token A with token B and vice versa. Suppose a pool consists of x token A and y token B. The price of assets is ruled by the constant product formula, meaning that, at each swap, the pool preserves $x * y$. When a user swaps a token A for token B (the user adds token A to the pool and takes token B from the pool), x increases by a and y decreases by b, where b is computed so that $x * y$ is constant. Thus, token A's value decreases while token B's increases, and the two parts maintain the same value. Uniswap implements the AMM model using mainly three smart contracts.

- The *Factory* contract creates the smart contract that handles liquidity pools. It is responsible for creating one and only one liquidity pool for each token pair. Each time it creates a new liquidity pool, the Factory contract emits a *PairCreated* event.
- The *Pair* contract implements the AMM logic and keeps track of the pool's status, including the token balances. The Pair contract emits three Events that notify the liquidity Pool's status changes. The Pair contract emits a Mint (or Burn) Event each time an LP-token is minted (or burned) and a Swap event each time a user swaps tokens in a liquidity pool. All liquidity pools created by the Factory smart contract implement these events.
- The *Router* offers an entry point to interact easily with the other Uniswap smart contracts. By interacting with the Router, it is possible to create liquidity pools, add and remove liquidity, and swap tokens.

The success of Uniswap's model and its open-source [60] nature has produced the emergence of hundreds [37, 38] of decentralized exchange (DEX) protocols across various blockchains. These protocols, called *forks*, leverage Uniswap's publicly available smart contracts as a foundation for their platforms. While Uniswap has evolved to its fourth version, most forks are based on Uniswap v2 due to its permissive open-source license. One of the most famous Uniswap forks is PancakeSwap [45]. PancakeSwap operates on the Binance Smart Chain, with over $2 billion[4] in Total Value Locked (TVL) within its smart contracts.

3 Sniper bots

Sniper bots are software applications that monitor a specific activity to automatically perform an action before anyone else [14]. Examples of these bots are "Scalping bots" [10], programs designed to purchase limited-availability goods quickly. These kinds of bots have been used to buy limited-edition sneakers [42] and Nvidia GPUs during the 2021 graphic card shortage [10]. The goal of bot users is usually to resell purchased items at a higher price. In the blockchain world, sniper bots are used to buy tokens as soon as they are listed on an AMM platform.

3.1 Sniper bots dataset

To understand how sniper bots are implemented and the features they offer, we leverage Github [28], one of the most popular Internet hosting services for software. We systematically search sniper bots on Github using keywords such as:

[4]Data retrieved from DefiLlama [36] in 2024-04-17

"Sniper bots," "Sniping bots," and other similar variations. In this way, we find hundreds of open-source repositories. To select the most popular and arguably the most used, we leverage GitHub's star ranking system [9]. In particular, we select repositories with at least 15 stars, accounting for 70 different sniper bots. Then, we discard from our analysis 25 repositories containing the code of sniper bots unrelated to AMMs. Most of these sniper bots are used in online video games (8 repositories) or to buy NFTs as quickly as possible in NFT marketplaces (5 repositories). Analyzing the remaining repositories, we notice that 17 of them do not contain open-source code. In these cases, the repositories are used to promote and sell closed-source sniper bots. Some others contain only executable files and instructions on how to use the bots. In the end, considering only the open-source implementations, we focus on analyzing 28 repositories.

Analyzing the sniper bots' source code, we first notice that almost all the considered sniper bots target Ethereum or the BNB Smart Chain (BSC). The only exception is a sniper bot that operates on the Avalanche [54] blockchain. In particular, three sniper bots exclusively support Ethereum, 17 support the BSC, and seven offer multi-chain support, being able to target both Ethereum and BSC. Looking at the implementations in more detail, we find these bots target the PancakeSwap and Uniswap AMMs. Only a few of them also offer the possibility to snipe tokens released on other AMMs operating on the Ethereum and BSC blockchains. Analyzing the code repositories, we find that there are two categories of sniper bots:

- **Single-target sniper bots.** These sniper bots target a specific token, requiring the user to input the smart contract address of the token. A user can use this kind of sniper bot to buy the token of a highly hyped project at its listing price, expecting its value to skyrocket right after. We find 25 implementations of this kind of sniper bot.
- **Multi-target sniper bots.** The second category of sniper bots is designed to buy every token as soon as it is listed. In this case, the goal of their users is more speculative. Indeed, the strategy behind these bots is to buy as many different tokens as possible, hoping that at least a few will gain value in the future. In our dataset, we find three implementations of multi-target sniper bots.

3.2 The anatomy of sniper bots

Analyzing the code repositories reveals that, despite having different goals, these two categories operate similarly and follow the same execution phases, which we illustrate in Fig. 2 and report in the following.

(1) **Find the liquidity pool.** As a first step, the sniper bot must identify the liquidity pool from where to buy the target token. Since these bots aim to buy the token as soon as it is listed, the sniper bot looks for newly created liquidity pools available for trading (i.e., actually containing liquidity). In § 3.2.1, we will explore the strategies the sniper bots implement to handle this phase.
(2) **Scam protection.** Then, the sniper bot can perform some checks to ensure the token to buy is not a scam. If these checks fail, the sniper bot will not buy the token, and in the case of a multi-target sniper bot, it will search for a new liquidity pool. This kind of security measure is implemented only by 13 sniper bots. We will explore the different implemented scam check solutions in § 3.2.2.
(3) **Advanced features.** Since some token smart contracts implement techniques to avoid bot interactions (§6), the sniper bot can implement workarounds to evade detection and still buy the token. We find advanced features implemented in 10 sniper bots. We will describe the anti-bot mechanisms in § 3.2.3.
(4) **Buy the token.** Finally, the sniper bot buys the desired amount of the token. This phase can be performed by interacting with the AMM router, performing a simple swap operation, or interacting with a custom smart contract. We will explain these two techniques in § 3.2.4.

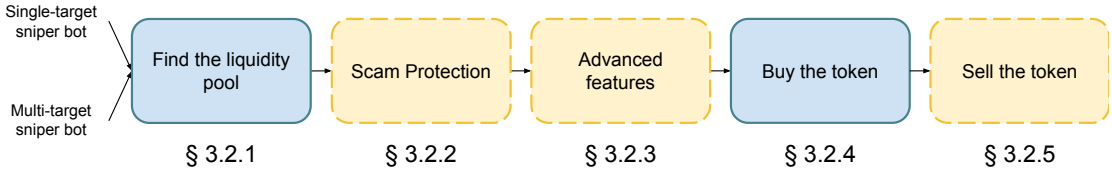

Fig. 2. The phases of a sniper bot's execution. We report in blue the phases we always find implemented by sniper bots. Instead, we report in yellow the optional phases that a sniper bot can implement to improve its usability.

(5) **Sell the token.** The sniper bot can also offer the possibility of selling the token automatically. In § 3.2.5, we describe how this phase is implemented by the 10 sniper bots that offer this feature.

3.2.1 Find the liquidity pool. Exploring the source code of the sniper bots in our dataset, we find that they use different methodologies to discover new liquidity pools. In particular, we find that single-target sniper bots use the following techniques:

Mempool scan. The fastest way to find a liquidity pool for the target token as soon as it is available is to monitor the blockchain mempool (§ 2.1.1)—the list of pending transactions waiting to be included in one of the next blockchain blocks. In this case, the sniper bot monitors the mempool, searching for the first transaction that adds liquidity to the target token's liquidity pool. Technically, this is done by checking the bytecode of the transaction for the signature of the *addLiquidity* function of the Uniswap Router (i.e., the function used to add liquidity to a liquidity pool).

Leverage Uniswap smart contracts. A single-target sniper bot directly interacts with the AMM smart contracts. It calls the *getPair* function of Uniswap's Factory contract at regular intervals. This function takes a pair of token addresses as input. It returns the address of the liquidity pool that contains the pair if it exists or the zero address if it does not. Thus, the sniper bot can use *getPair* providing as input the token to snipe and the coin used to buy it (e.g., ETH). This method is one block slower than the previous one. Indeed, for the Uniswap smart contract to be updated, the transaction that updates its status must be confirmed in the blockchain. On average, a new block is generated every 13 seconds on Ethereum, while on BSC every 3 seconds. However, even if the liquidity pool exists, it is not guaranteed to contain liquidity. Indeed, a user can create a liquidity pool but not add tokens to it, making any swap impossible. Thus, to understand if there is liquidity, the sniper bot performs polling requests to the *getReserves* function of the contract. This function returns the quantity of the two tokens in the pool. The sniper bot performs the swap only when this quantity differs from zero.

Instead, we find that multi-target sniper bots usually follow one of these approaches:

Event Log monitoring. Sniper bots monitor the blockchain Event Log looking for new *PairCreated* events. As mentioned in § 2.3, this Event is emitted by the Factory contract of Uniswap each time it creates a new liquidity pool. From the data in this Event, the sniper bot can retrieve the addresses of the two tokens in the liquidity pool and the address of the liquidity pool itself. As for the previous case, the sniper bot must verify that the liquidity pool contains the tokens. Thus, before sniping the liquidity pool, it ensures that there is liquidity through the *getReserves* function.

Telegram channels. Some sniper bots use Telegram [56], a very popular messaging app with more than 900 million active users, as a source to discover new liquidity pools. Indeed, token developers often announce the release of their tokens on Telegram channels[5] dedicated to token release announcements. These sniper bots use Telegram APIs to monitor a list of channels. The sniper bot parses newly published messages from these channels, looking for the address

[5]Channels are public groups that anyone can join but where only the admin can write [33]

of a liquidity pool created on the target AMM. For instance, we find a sniper bot that monitors Telegram channels[6] reporting newly listed tokens by CoinMarketCap [17] and CoinGecko [16], two of the most popular cryptocurrency aggregator websites. Usually, the list of monitored channels is customizable by the user, which can add or remove specific channels. Some sniper bots suggest monitoring specific channels. Additionally, users can specify a list of token addresses or words blocklisted to avoid buying specific tokens or tokens including in their name specific words.

3.2.2 Scam protection. Sniper bots often perform checks to avoid buying scams or suspicious tokens. This is not surprising, as anecdotal evidence (e.g., SquidGame [49]) and previous works [14, 41, 62] have shown that investing in liquidity pools can be risky as thousands of tokens are purposely created to perform scams. The scam protection features are mainly designed to prevent two threats: Rug pulls [14, 62] and honeypot tokens [27, 59]. Rug pulls are commonly known scams, where a development team abandons a project and runs away with investors' funds [62]. In the context of AMMs, a malicious actor creates a scam token and adds it to a liquidity pool with a valuable token (e.g., ETH). Then, he waits for someone who buys the scam token, swapping it with the ETH. Once the scammer has reached their desired profit, he drains the pool, taking all the invested ETH and leaving investors with a worthless token. Instead, a honeypot is a token whose smart contract has been altered to modify the standard transfer functionality. In particular, these tokens usually contain a modified *transferFrom* function that Uniswap uses to perform the buy and sell operations in the liquidity pool. The modified code can prevent the token from being sold or impose very high fees for sell operations.

We find that sniper bots employ the following solutions to avoid these threats:

Trial trade. A possible countermeasure to avoid falling prey to honeypots is to perform a trial trade. With this practice, the sniper bot buys a small number of tokens and right after sells them. This practice aims to check that the token smart contract does not prevent the sale of the token. Thus, if the trial trade is successful, the sniper bot purchases the desired token amount.

RugDoc. A second possibility is leveraging the APIs of RugDoc [51], a tool designed to help DeFi investors to make informed decisions about the tokens they choose to invest in. RugDoc performs some tests on the token to check if it is a honeypot and provides results through APIs. So, the sniper bot queries the RugDoc's APIs to retrieve the tests' results and infer the risk level of the target token. If the estimated level of risk is acceptable, the sniper bot buys the token.

Source code check. Before buying the token, some sniper bots check the source code of the smart contract. In particular, they only buy tokens whose smart contract is public and verified on popular blockchain explorers like Etherscan [22] (for Ethereum) or BscScan [11] (for BSC). These websites offer contract verification where developers can publish their smart contract source code on the site. The site will then compile the code and check if the generated bytecode matches the stored bytecode on the blockchain. If it matches, the contract is considered verified. Other than the verified status, we find sniper bots that avoid buying the token if the smart contract contains specific keywords, such as: "HelloBEP20", "BEP20TOKEN", "blacklist", or "allowed". Indeed, according to the sniper bots' documentation, the first two keywords may indicate that the smart contract has been generated with a tool and, thus, it is a project not to trust. Instead, the latter two keywords might indicate the presence of functionality in the smart contract that can restrict token trading.

Liquidity check. Lastly, some sniper bots offer the feature to buy only in liquidity pools with more than a certain amount of liquidity. Sniper bots perform this check by calling the *getReserves* function of the liquidity pool's smart contract.

[6] https://t.me/CMC_CG_listing_alerts, https://t.me/CMC_fastest_alerts

3.2.3 Advanced features. Some sniper bots offer advanced features to circumvent smart contract functionalities designed to directly or indirectly limit the action of sniper bots. Indeed, as we will see in Sec. 6, several token smart contracts implement techniques to hinder sniper bots or bots in general (e.g., trading bots). In the following, we report the advanced features we find implemented by sniper bots to evade anti-bot countermeasures.

Wait *n*-blocks. This feature allows the user to specify the number of blocks the sniper bot waits to purchase the token after the liquidity is added. This precaution is used to avoid penalties imposed by some token smart contracts that want to penalize automatic trading actions at the early stages of the liquidity pool. For instance, a smart contract may blocklist addresses that purchase the token in the same block of its launch, preventing those addresses from making any future token transfers. This countermeasure is described in detail in the *Sniper bots blocklist* paragraph in Sec. 6.1. Other smart contracts may impose a very high fee on purchase transactions made within the first few blocks after liquidity is added (*e.g.*, 99% of the acquired tokens are returned to the liquidity pool). This is an example of the *Tax during launch window* countermeasure illustrated in Sec. 6.1.

Check trading enabled. As explained later in the *Disabled trading* paragraph of Sec. 6.1, some token smart contracts present the functionality to enable or disable token transfers at any time. The token creator can use this functionality for different technical or marketing reasons. For instance, the creator could set up the liquidity pool and launch the token on the market some days later. Thus, once the liquidity pool is created, the token creator will disable the token transfer, and hence the trading, to enable it on launch day.

Most sniper bots in our dataset are implemented to buy the token as soon as liquidity is added to the pool. However, if token transfers are disabled, the buy transaction will fail, and the bot typically will not attempt to repurchase the token. To handle this case, some sniper bots implement methods to detect when transfers are enabled. One approach is sending a small test transaction; if successful, the bot follows with a second transaction to buy the desired amount. Additionally, we found two additional strategies in our dataset. In the first, the bot polls the liquidity pool's smart contract, monitoring price changes. If the price fluctuates, the bot infers that transfers are enabled and attempts to buy the token. In the second approach, the bot monitors the mempool for transactions containing bytecode of known functions used to enable token transfers, such as *openTrade*, *enableTrading*, and *tradingStatus*. These two strategies are more efficient than the first, where the sniper bot still pays the transaction fee if the small test transaction fails.

Multiple buys. There are smart contracts that restrict the number of tokens the same address can buy in a single transaction. This countermeasure, described in the *Token amount limit* paragraph of Sec. 6.1, prevents big players—also known as *whales*— from buying a large token supply in a short amount of time. Even if not intended to contrast sniper bots directly, this mechanism can cause the sniper bots' buy transactions to fail if the desired quantity of tokens overcomes the restriction of the smart contract. Some sniper bots can buy the desired amount of tokens using multiple buy transactions, working around the smart contract limitation.

In Sec. 6, we will analyze the code of public smart contracts to quantify the diffusion of practices that can hinder the sniper bot operations.

3.2.4 Buy the token. Finally, the sniper bot buys the token. In particular, we find two ways the sniper bots perform the purchase:

Interacting with the Router contract. The sniper bot can buy the token by sending a transaction to the Router contract of the target AMM. To finalize the purchase, the user of the sniper bot has to specify the number of tokens to buy and the maximum *slippage* (i.e., the difference between the expected and the actual price) tolerated. Specifically, the bot specifies the amount of cryptocurrency to spend and calculates the estimated amount of tokens to receive in return.

Table 1. Summary of sniper bot features.

Feature	Number of Sniper Bots	Feature	Number of Sniper Bots
Find the liquidity pool (Sec. 3.2.1)		*Scam protection (Sec. 3.2.2)*	
Mempool scan	16	Trial trade	1
Leverage Uniswap smart contracts	3	RugDoc	3
Event Log monitoring	9	Source code check	4
Telegram channels	3	Liquidity check	8
Advanced features (Sec. 3.2.3)		*Sell the token (Sec. 3.2.5)*	
Wait n-blocks	9	Sell percent gain	13
Check trading enabled	4	Stop loss	10
Multiple buys	9	Trailing stop	9

However, the token price can change due to other swap transactions before the execution of the sniper bot's trade. For this reason, there can be a difference between the estimated and actual token amounts. To account for this, most sniper bots have a slippage setting, which allows traders to specify the acceptable difference between the expected and executed trade price. High slippage values maximize the chances of successfully buying the token, while a low slippage offers greater assurance on the number of tokens received.

Using a custom smart contract. The sniper bot buys the token by sending a transaction to a custom smart contract rather than directly to the AMM router. This approach incurs higher costs, including smart contract deployment fees, but provides advantages. The advantage is that the custom smart contract enables atomic execution of multiple operations, such as checking if the token is a honeypot.

3.2.5 Sell the token. While all sniper bots are used to buy tokens, not all of them offer an automatic way to sell. Indeed, the selling functionalities are present only in 10 out of 28 sniper bots.

Sell percent gain. The sniper bots that automatically sell tokens allow the user to set a target profit percentage. Once the token's value increases by the designated percentage, the sniper bot automatically sends a swap transaction to sell the token.

Stop loss. Most sniper bots also provide a mechanism to protect investors from excessive loss, namely a *stop loss*. The stop loss is a simple threshold and allows the bot to sell the tokens if the token price drops below a specified percentage relative to the buy price.

Trailing stop. Some sniper bots implement a more sophisticated trading strategy known as *trailing stop*. With the trailing stop, the sniper bot continuously tracks the token's price. If the maximum value assumed by the token falls below a given percentage, the sniper bot automatically sells the token. The trailing stop is more adaptable than a fixed stop loss or selling at a fixed profit target. It enables the sniper bot to follow the token's price trend until it changes direction, potentially increasing profits.

Tab. 1 reports the number of sniper bots that present specific features.

4 Sniper bots detection

In the previous section, we were interested in understanding how sniper bots work by analyzing their source code. In this section, we change perspective, investigating how they are operatively used by analyzing blockchain data.

Table 2. Summary of the Liquidity Pools dataset.

	Ethereum	BSC
PairCreated	155,530	1,441,911
Mint	3,151,983	40,623,003
Swap	114,825,502	1,102,575,865
Tokens	144,145	1,250,913

4.1 Liquidity pools dataset

To study the sniper bots, we create the liquidity pools dataset, a collection of liquidity pools and their operations on Ethereum and BSC, the two blockchains most targeted by the sniper bots found on GitHub. Given the large quantity of data to retrieve, we run an Ethereum and a BNB Smart Chain node on our machine and synchronize the two blockchains. Then, we use Web3 [48], a Python library that allows interaction with EVM-compliant nodes to query the blockchains and obtain the data from their inception to February 2023. To collect the data, we use the same approach of previous works [14, 41, 62]. In particular, we parse the Event Logs of both blockchains, collecting Events that comply with the Uniswap smart contract implementation. Note that all Uniswap forks, including those deployed in the BSC, also implement these Events. In detail, we retrieve the data of the following events: PairCreated, Mint, and Swap.

- **PairCreated:** With this Event, we collect the addresses of liquidity pools and other relevant data: the addresses of the two tokens they contain, the block number where the pool was created, the transaction hash, and the address that created the pool. We find 155,530 liquidity pools on Ethereum and 1,441,911 on BSC that contain in their pairs 144,145 unique tokens on Ethereum and 1,250,913 unique tokens on BSC.
- **Mint:** Collecting Mint events, we infer when liquidity providers added liquidity to the pool. From the Mint, we collect the address that added the liquidity, the amount of liquidity added, the address of the pool, the transaction hash, and the block where the operation occurred. We collect 3,151,983 Mint events on Ethereum and 40,623,003 Mint events on BSC.
- **Swap:** Gathering Swap Events, we obtain information such as the transaction hash, the block in which the operation occurs, the address that performs the swap, the address of the liquidity pool, the number of tokens swapped, the gas used, and the gas price. We collect 114,825,502 Swap events on Ethereum and 1,102,575,865 Swap events on BSC.

Tab. 2 succinctly reports the events we collect and the number of tokens in liquidity pools.

4.2 Sniper bots identification

As a first step towards understanding how sniper bots are operatively used, we must identify their operations by analyzing their on-chain transactions. Leveraging the idea that sniper bots are faster than humans and aim to buy newly listed tokens as soon as possible, to identify them, we consider only the transactions immediately following a token's listing. Indeed, it is extremely unlikely that a human can perform operations on the liquidity pool in the very few seconds after the listing, as one would need to manually inspect the mempool or the last mined block and swiftly send a transaction with an appropriate gas price. Thus, for each liquidity pool, we retrieve all transactions interacting with it that occur in the same block of its first Mint event (i.e., the liquidity was added for the first time) and those in the following block. For the remaining sections, we refer to the block where the first addition of liquidity occurs as *Block-0*, and as *Block-1* the block immediately following. By retrieving transactions at Block-0, we aim to select all

Table 3. Summary of the Sniper Bot operations dataset.

	Ethereum	BSC
Number of Transactions	322,344	1,491,767
Number of Liquidity Pools	49,422	507,312
Number of Buys	352,413	1,716,917
Number of Sells	34,222	240,579

swaps made by sniper bots that monitor the mempool, while, with those at Block-1, transactions made by sniper bots that read the Event Log or that interact with Uniswap's smart contracts (see Sec. 3.2). Furthermore, since liquidity pools can consist of any pair of ERC-20 (or BEP-20) tokens, it can be challenging to identify what token is the target of the sniper bot. However, when the liquidity pool consists of the native coin of the blockchain (in our case, ETH or BNB) and an ERC-20 (or BEP-20) token, we can confidently infer that the target of the sniper bot is the ERC-20 (or BEP-20) token within the pair. So, although sniper bots can target any liquidity pool, we analyze only those consisting of the native coin of the blockchain and an ERC-20 (BEP-20) token. These represent 91.6% (137,144) and 88.4% (1,086,258) of the liquidity pools on Ethereum and BSC, respectively. Narrowing our research on these liquidity pools allows us to define two operations: the *buy* and the *sell*. In particular, we define as a buy operation any swap that takes as input ETH (BNB) and provides as output any other ERC-20 (BEP-20) token. Instead, we define a sell operation as any swap that takes an ERC-20 (BEP-20) token as input and provides as output ETH (BNB). Then, we polish our dataset, discarding operations we can safely assume are not linked to sniper bot activities. In particular:

- We remove all the buy operations performed in the liquidity pool created by the same address performing the swap (1,053 cases). Indeed, given the speculative nature of sniper bots, we can safely assume that the user of a sniper bot never targets a token that he created.
- We eliminate all buy and sell operations associated with well-known MEV activities. Indeed, by analyzing the buy operations executed in the first two blocks following a token release, we may inadvertently detect the operations of MEV bots that exploit sniper bot activities for profit. Therefore, to ensure that we only capture the operations of sniper bots, we have made a conservative choice to identify and discard any activities that exhibit patterns consistent with previously studied MEV bots. In particular, we remove all the operations under the sandwich attack definitions proposed in [50, 58] (4,304 cases).

Analyzing the transactions performed at Block-0 and Block-1 on the liquidity pools of our dataset, we find 322,344 transactions involving 49,422 liquidity pools on Ethereum and 1,491,767 transactions involving 507,312 liquidity pools on BSC. Ethereum transactions perform 352,413 buy and 34,222 sell operations. Meanwhile, those of BSC perform 1,716,917 buy operations and 240,579 sell operations. Tab. 3 describes the final dataset of sniper bot operations.

Note that, in both blockchains, the number of total operations (buys and sells) exceeds the number of transactions. Such discrepancy is due to sniper bots' features (see Sec. 3.2) allowing the possibility of buying the target token via multiple swaps. This approach bypasses restrictions imposed by smart contracts that limit the number of purchasable tokens in a single swap.

4.3 Selected transactions analysis

To confirm that our dataset contains transactions performed by sniper bots, we analyze some characteristics of the selected transactions. Sniper bots are designed to utilize gas prices identical to or higher than the target *AddLiquidity*

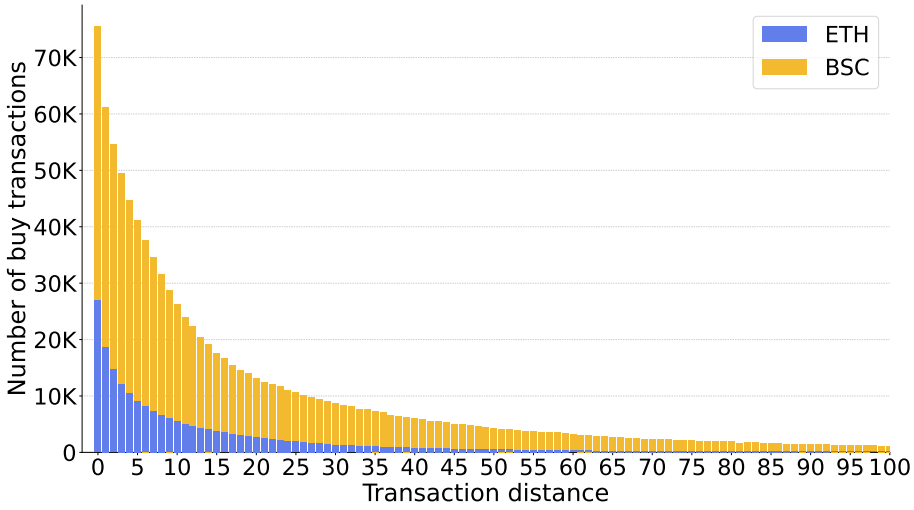

Fig. 3. Transaction distances of selected Ethereum and BSC buy transactions. For visualization, the chart reports the values with a transaction distance less than or equal to 100.

transaction. Strategically setting the gas price value increases the likelihood of executing the buy transaction immediately after the liquidity is added. This is important because being the first transaction after the liquidity pool became active guarantees the possibility of buying the tokens at the listing price. Meanwhile, subsequent swap operations by others result in the purchase of tokens at higher prices, thus reducing profit opportunities. Our analysis reveals that 90.9% of selected buy transactions use the same gas price as the associated *AddLiquidity*, while 5.3% and 3.8% exhibit higher and lower gas prices, respectively. We observe minimal average gas price differences between these two last categories. On average, transactions on Ethereum and BSC with higher gas prices use $6.8 * 10^{-8}$ ETH and $9.2 * 10^{-8}$ BNB more than their corresponding *AddLiquidity* transactions. Conversely, on average, transactions with lower gas prices use $3.1 * 10^{-8}$ ETH and $9.4 * 10^{-8}$ BNB less. These minimal gas price differences suggest that the senders of the transactions are aware of the gas price employed by the *AddLiquidity* transaction. Thus, they strategically adjust their gas price (slightly higher or lower) to include their buy transactions close to the *AddLiquidity*.

To better evaluate how close they are, we introduce the concept of *transaction distance*. We define transaction distance as the number of interleaving transactions between two of the same block. Thus, a zero distance means that the two transactions are written in the same block, one after the other. At the same time, a value x greater than zero indicates that there are x interleaving transactions that separate them. Fig. 3 shows the number of transactions at a specific transaction distance. Across both blockchains, a significant portion of transactions exhibit a transaction distance of zero. Specifically, 26,940 (8.3%) Ethereum transactions and 75,525 (5.1%) BSC transactions have a transaction distance of zero. Examining transactions with a transaction distance greater than zero, we discover that most selected Ethereum transactions (51.3%) have a transaction distance of less than nine relative to the *AddLiquidity* transaction. Similarly, on BSC, 51% of the selected transactions have a transaction distance below 13. These findings suggest that the selected transactions exhibit characteristics that are highly consistent with those typically associated with transactions sent by sniper bots.

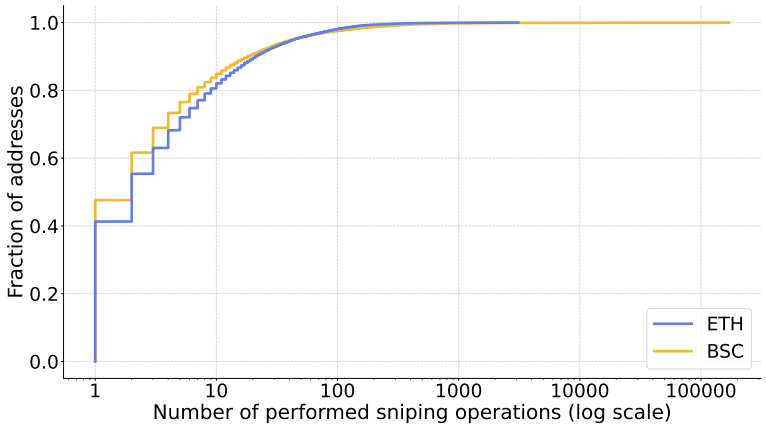

Fig. 4. The distribution of the number of sniping operations performed by Ethereum and BSC addresses.

5 Results

In the following section, we analyze the swaps conducted by sniper bots at Block-0 and Block-1. Specifically, we study the methodologies used by the sniper bots. Then, we evaluate the impact of sniper bot operations on the token market to understand how token prices fluctuate due to purchases made by sniper bots. Finally, we quantify the gains and losses experienced by sniper bot users.

5.1 Sniper bots on Ethereum and BSC.

Sniper bots prefer BSC. As we saw in the previous section, by analyzing the number of sniping operations performed in the liquidity pools of our dataset, we find that on BSC (1,491,767), there are approximately five times the number of sniping operations than on Ethereum (322,344). Looking at the number of addresses using the sniper bot, we find that the number of addresses that perform at least one sniping operation on the BSC (44,471) is almost double that on Ethereum (27,691). Finally, observing all the liquidity pools in our dataset, it is possible to note that on Ethereum, 31.7% (49,422 out of 155,530) of them experienced a sniping operation, while on BSC, 35.1% (507,312 out of 1,441,911).

A small number of addresses engage in a lot of sniping activity. Studying the number of sniping operations performed by individual addresses, we find that the distribution on the two blockchains is very similar. Fig. 4 shows the CDF of the number of sniping operations performed by the addresses.

Both distributions show very low sniping activity per address, with the BSC addresses exhibiting a slightly lower tendency to snipe than Ethereum. Specifically, 54.4% of Ethereum addresses perform at most two sniping operations, while this figure rises to 67.3% for BSC addresses. Furthermore, most (over 80%) of the addresses on both blockchains perform less than ten sniping operations (82.2% for Ethereum and 84.8% for BSC).

Interestingly, a small subset of addresses is responsible for a very high percentage of sniping operations in both blockchains. Indeed, considering only the top 1% of Ethereum addresses, they account for 30.8% of the total sniping operations performed on the blockchain. On the other hand, on BSC, the top 1% of addresses are responsible for 70.2% of sniping operations. The addresses of these two subsets carry out sniping operations in at least 147 liquidity pools on Ethereum and 271 on BSC. Among the addresses of these subsets, we find cases that stand out. On Ethereum, one

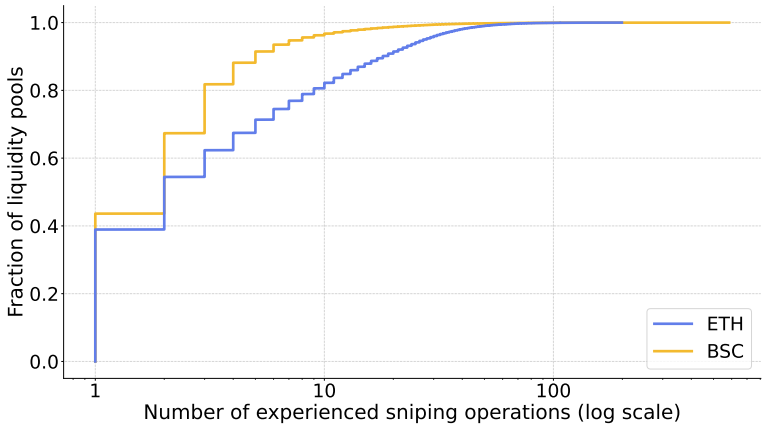

Fig. 5. The number of sniping operations performed inside the liquidity pools of Ethereum and BSC.

address[7] is responsible for 3,100 sniping operations, while on BSC, another[8] is responsible for 170,450 operations. Both addresses perform each sniping operation in a different liquidity pool.

BSC liquidity pools experience fewer sniping operations than Ethereum. In this analysis, we change perspective, switching from the point of view of the addresses to those of the liquidity pools. In particular, we analyze how many sniping operations each liquidity pool experiences. Fig. 5 displays the number of experienced sniping operations by the fraction of liquidity pools targeted by at least one sniper bot. Interestingly, despite a higher prevalence of sniper bots on the BSC, Ethereum liquidity pools targeted by these bots experience a noticeably more significant number of sniping operations. As we can see, almost all of the liquidity pools (96.7%) on BSC suffered less than ten sniping operations. Meanwhile, on Ethereum, this percentage is 82.2%. In light of these results and the analysis we perform in Sec. 5.2, we can argue that users leveraging sniper bots in Ethereum are more selective regarding the coin to target. A possible reason for that could be the most expensive transaction fees of the Ethereum network, which lead the user to be more selective about the tokens to snipe.

5.2 The economic impact of sniper bots

As shown in the previous section, sniper bots perform a relevant number of sniping operations on Ethereum and BSC. In this section, we quantify their investments and their impact on the price of the newly listed tokens.

Sniper bots buy tokens for hundreds of millions of US dollars. As the first step toward understanding the economic impact of the sniper bots, we compute the amount of money these bots invest in the market. To do so, we sum the amount of ETH (BNB) they swap into the liquidity pools at Block-0 and Block-1. Summing up the buy operations, we observe that sniper bots have a significant economic impact. Indeed, globally, sniper bots bought tokens for $155,630,184.2 (99,036.3 ETH) on Ethereum and $137,548,859.7 (335,696.2 BNB) on the BSC.

Sniper bots on Ethereum invest significantly more than on BSC. Analysis of individual sniping operations unveils a significant disparity in investment amounts between Ethereum and BNB Smart Chain (BSC) sniper bots. Ethereum operations exhibit a markedly higher average investment of $489.1 compared to $92.5 on BSC. This contrast is also

[7] `0x00ABC123bAc9cC2Cea5Bf0CFbf622f781eFa897c`
[8] `0xE71A69F434A719Cd5AFa411568B69716aaB3B84a`

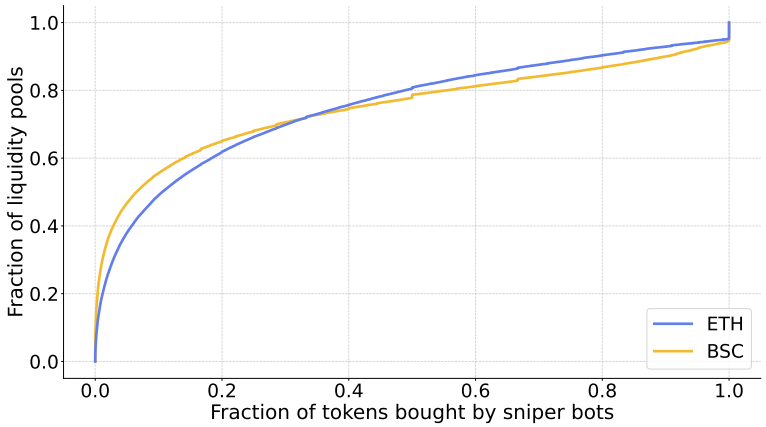

Fig. 6. Fraction of tokens bought through sniping operation w.r.t. to the total token bought inside a liquidity pool.

evident in the overall volume of tokens acquired on the respective blockchains. On average, sniping activities on Ethereum liquidity pools resulted in the acquisition of significantly more tokens, with an average value of $3,167. This value shrinks drastically on BSC, where the average is only $271.8. The higher operational costs on Ethereum likely explain this divergence in investment. Indeed, by analyzing transaction fees paid by sniper bots, we observe a significant disparity: the average cost to execute a sniping operation is $29.60 on Ethereum, compared to just $0.60 on BSC.

Sometimes, only sniper bots buy newly listed tokens. Analyzing the life of the liquidity pools targeted by sniper bots, we discover that sniper bots buy a significant portion of the newly listed tokens. Fig. 6 shows the fraction of tokens purchased by sniper bots in each liquidity pool. On average, sniping operations buy 24.8% and 25% of the total purchased tokens in the liquidity pools of Ethereum and BSC, respectively. Furthermore, for the 19.4% of Ethereum liquidity pools and 21.8% of BSC, the amount of tokens acquired by sniper bots represents the majority of the investments made. Surprisingly, in 4.6% of Ethereum liquidity pools and 5% of BSC, sniper bots are the only ones that perform a swap to buy the new tokens. There could be multiple reasons causing sniper bots to be the only owners of specific tokens. Firstly, these tokens, such as meme coins or those created solely for network testing, may not have fundamental value. Consequently, traditional investors may be less interested, making the sniper bots the only buyers. The second is that the liquidity pool was active for a very short time, and thus, no other investor was fast enough to buy the new token. Usually, this is due to a market manipulation known as Rug Pull. Indeed, Cernera et al. [14] showed that almost 60% of Ethereum and BSC liquidity pools have a duration of less than 24 hours before the liquidity pool creator removes all the liquidity, consequently making impossible to perform any further swap. To confirm this hypothesis, we replicate the analysis of Cernera et al. to identify liquidity pools that suffer a rug pull in the first 24 hours of their life. At the end of the analysis, we discover that among the liquidity pools in which the swaps are performed only by sniper bots, 45.3% (1,008) of them on Ethereum and 60.1% (14,856) on BSC experienced a rug pull. Moreover, these liquidity pools were active for an incredibly short time: on average, 89.9 minutes for Ethereum liquidity pools and 69.4 minutes for BSC.

Token prices fluctuate sharply due to sniper bot activity. Having quantified the token amounts acquired by sniper bots, we want to understand their impact on the tokens' price. To this end, we start by determining the token's listing price. Recall that the ratio of the token amounts in the liquidity pool gives the price of the two tokens. So, by parsing the information in the transaction that injects the first liquidity (AddLiquidity) into the liquidity pools, we figure out the

amounts of the two tokens and their listing price accordingly. Then, we compute all the transactions recorded at Block-0 and Block-1, updating the reserve of both tokens accordingly to the swap performed into the liquidity pool. After this procedure, we are able to estimate precisely the amount of both the tokens within the liquidity pool and, therefore, the pricing of the new token at the end of Block-1. Comparing the price of the new token at the end of Block-1 with that established by the initial injection of liquidity, we discover that sniping operations significantly affect token prices.

As we can expect, because of the higher amount of money put in the line on Ethereum, tokens on this blockchain suffer a price increase significantly higher than those on BSC. In particular, the median price increase on Ethereum is 19.7%, while on BSC, it is 4.4%. We notice an even more marked difference in the 75^{th} percentile and the maximum values. Indeed, on Ethereum, the 75^{th} percentile and the maximum value accounts for 72.3% and 172.9%, respectively, while on BSC for 31.9% and 79%.

After examining how sniper bot purchases influence the prices of target tokens, we shift our analysis to the impact of their sell operations. Before conducting our study, we exclude sale operations that occur after liquidity pools are targeted by rug pulls. Indeed, the liquidity remaining in these pools is typically very thin, which can result in significant overestimation or underestimation of price variations. We analyze the complete history of transactions in the remaining pools, monitoring token prices before and after sale transactions to quantify their variation. Our analysis shows that sell operations result in an average price decrease of 5.1% on Ethereum and 7.4% on BSC. Interestingly, in both blockchains, a significant percentage of sell operations (3.6% on Ethereum and 8.2% on BSC) cause a decrease in the price of the token by more than 20%. These drops are even more significant considering that we estimated the price variation for each singular sell transaction executed by the sniper bot. Thus, if multiple sell transactions for the same token happen in a short period, they could cause a more significant price drop than we reported. Alternatively, if the sell transactions are spread out over a longer period, they can have a larger impact on the volatility of the token.

In conclusion, our findings suggest that sniper bots significantly impact the price of target tokens through their buy and sales operations. This behavior contributes to a fluctuating market that can hinder the reputation of tokens and discourage potential investors.

5.3 Gains

In this subsection, we analyze the operations carried out by sniper bots to estimate their profitability. For each sniping operation, we estimate its profit using the following formula:

$$balance = T_{out} - T_{in} - fees \tag{1}$$

T_{out} is the amount obtained by the sell operations, T_{in} is the amount spent to buy the token, and *fees* is the amount of transaction fees paid to perform the buy and the sell operations. In the following, we divide the operations into successful and unsuccessful, considering an operation successful if the *balance* is strictly positive. Following formula 1, we calculate the returns deriving from individual sniping operations, finding that on Ethereum, more than half of the operations, 56.1% (129,587), closed with a positive balance. In contrast, on BSC, just 37.5% (316,059) of sniping operations are successful. Furthermore, operations performed on Ethereum generally bring higher returns than those on BSC. Indeed, sniping operations on Ethereum lead to an average profit of $1,654.9, while on BSC of $480.6. Furthermore, as we can expect from the previous result, Ethereum's sniper bots are more profitable, generating $214,456,832.6 (101,735.2 ETH) in total profits compared to BSC's $151,922,753.2 (268,374.9 BNB). Interestingly, losses from sniping activity were similar on both chains. Ethereum users lost $25,834,191.6 (13,215.4 ETH), while BSC users lost $31,901,324.7 (78,632.5

Table 4. An overview of the economic results of sniper bots.

Metric	Ethereum	BSC
# Successful Sniping Operations	129,587	316,059
# Unsuccessful Sniping Operations	101,318	528,053
Tot. Gain	$214,456,832.6	$151,922,753.2
Tot. Loss	$25,834,191.6	$31,901,324.7
Max. Gain	$6,400,007.9	$3,223,042.3
Max. Loss	$803,946.1	$208,723.3
Avg. Gain	$1,654.9	$480.6
Avg. Loss	$254.9	$60.4

BNB). Finally, an Ethereum unsuccessful operation results in an average loss of $254.9, while on BSC of $60.4. These results indicate that while a sniping operation on Ethereum brings, on average, higher earnings than one performed on the BSC, it also involves a greater risk of losses. Tab. 4 summarizes the gain and loss values of the sniper bots.

In the following, we analyze unsuccessful operations to understand better the reasons behind the high difference in success rate between Ethereum and BSC. We first discovered that 43.2% (43,850 out of 101,318) of unsuccessful operations on Ethereum and 14.9% (78,680 out of 528,053) on BSC closed with a negative balance because of the transaction fees. This higher percentage of unsuccessful operations on Ethereum is not surprising, considering the higher network fees. Instead, unexpectedly, we discovered that 2% (2,027) of the Ethereum unsuccessful sniping operations and 30% (158,416) of BSC are unsuccessful because they did not sell any of the acquired tokens. As discussed in SubSec 5.2, the liquidity pools for these tokens have been active for a very short period. Thus, as before, we check evidence of rug pulls for these liquidity pools. In this case, we discover that approximately 6.4% (131) of the operations on Ethereum and approximately 29.6% (47,032) on BSC fall prey to a rug pull. Therefore, sniper bots had no chance to sell the target token and get a profit. We focus our study only on the operations performed at Block-0 and Block-1, but sniping operations can also occur in later blocks. Thus, our estimation should be considered a lower bound of the phenomenon.

5.4 MultiChainCapital: a case study

This section provides a detailed example of a particularly successful sniping operation. The operation occurred on the Uniswap liquidity pool composed of the MultiChainCapital (MCC)[9] token and Ethereum (ETH). At block `13646960` (2021-11-19 05:46:12 PM UTC), liquidity is added to the liquidity pool for the first time. Specifically, the creator of the liquidity pool adds 1,000,000,000,000 MCC and 7 ETH. In the same block, at zero distance, an address[10] performs a swap purchasing 60,232,391,104.4 MCC in exchange for 0.5 ETH ($1,991.5). Subsequently, at block `13657656` (2021-11-21-2021 10:36:43 AM UTC), the same address executes a swap to sell 1,428,289,102.9 MCC in exchange for 10.5 ETH. By selling just 2.3% of the purchased tokens, it has already made a profit of 10 ETH ($45,844.8), equal to 2000% of the initial investment. Later, the address executes another ten transactions to sell all the remaining MCC tokens, executing the last sale at block `13688170` (2021-11-26 06:26:43 AM UTC). Thus, estimating the final balance of this operation with formula 1, the address investing 0.5 ETH in buying the token and 0.4 ETH in transaction fees, closed the operation with a net balance of 295.7 ETH ($1,322,939.7), making a profit of 59,140% in just one week.

[9]Smart contract address: `0x1a7981D87E3b6a95c1516EB820E223fE979896b3`
[10]`0x3FEbf8c5F93C1888B96bF13F5A8667e71C631244`

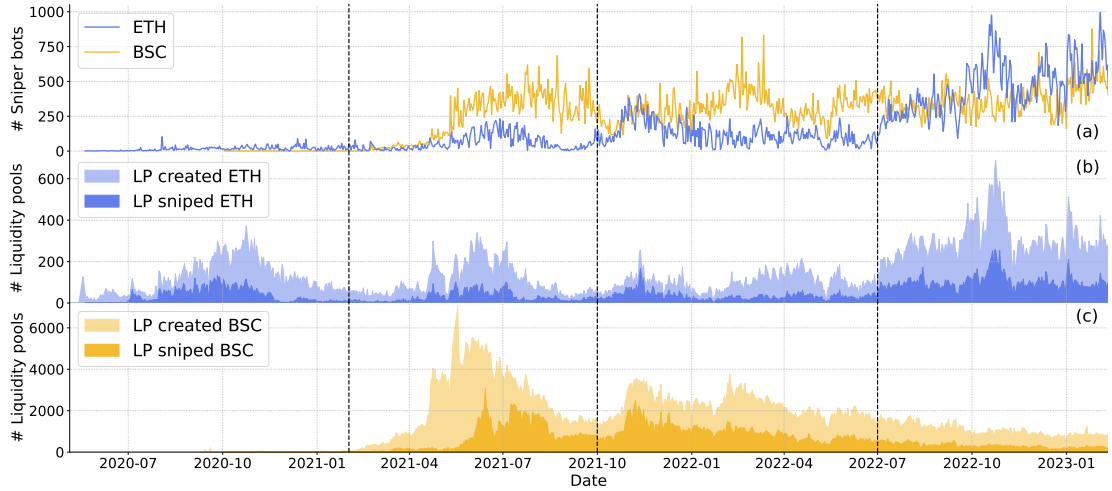

Fig. 7. The figure shows the number of sniper bots (a) and the number of Ethereum (b) and BNB Smart Chain (c) liquidity pools that have been targeted by sniping operations. All the metrics are aggregated daily. The dashed vertical lines divide the four phases we identify.

5.5 Longitudinal analysis

In this section, we analyze how the phenomenon of sniper bots evolved over time. Fig. 7 (a) reports the daily number of active sniper bots (Ethereum (in blue) and BSC (in yellow)), whereas Fig. 7 (b) and (c) the daily number of liquidity pools targeted by sniper bots over the number of liquidity pools created daily.

For Ethereum, the data span from May 20, 2020, the day of the first sniping operation on Ethereum, — two days after the release of Uniswap V2 — up until March 2023. Initially, the data relates exclusively to Ethereum, as the BSC had yet to be born. The data relating to the BSC sniper bots begins on October 3, 2020, only a few days after the launch of PancakeSwap (September 22, 2020), up to March 2023.

Analyzing the data reported in Fig. 7 (a), it is possible to note how the number of sniper bots per day and the number of tokens purchased are not constant. Indeed, we identify periods in which the activity of the sniper bots has intensified.
Phase 1: DeFi Summer. The first of these periods (July 2020 - February 2021) mainly concerns only Ethereum. These months align with the period commonly referred to as the *DeFi Summer 2020* [40]. During this period, the level of interest in DeFi has increased considerably, attracting new users looking for investment opportunities. Driven by the surge in the adoption and usage of DeFi platforms, it also increased the number of new tokens created and listed on liquidity pools (see Fig. 7 (b)). In this phase, the number of active sniper bots is pretty constant and limited (on average, 17.9 sniper bots daily). Instead, the percentage of target liquidity pools gradually increased, reaching a peak of 52.8% liquidity pool sniped on Sep 22, 2020.
Phase 2: The rise of BSC and the Altcoin Season. In the second phase (March 2021 - October 2021), the first DeFi protocols emerged on BSC, marking the beginning of its establishment as a cost-effective alternative to Ethereum. This attracted a significant influx of investors and capital to the platform. Moreover, this period aligns with the so-called Altcoin Season, a market cycle where alternative cryptocurrencies (all cryptocurrencies except Bitcoin, also called Altcoins) start to sharply increase their value and perform, from an economic point of view, better than Bitcoin. On one side, this phenomenon attracts new investors looking for easy gains; on the other, it stimulates the creation of tons of

speculative Altcoins. As we can see from Fig. 7, on Ethereum, the number of daily created liquidity pools triples during the Altcoin Season. Instead, on BSC, we recorded a growth of new daily liquidity pools from a few dozen to over 4,000, peaking at 7,086 (2021-05-18). Regarding the daily active sniper bots, we notice that they follow the same trends as the newly created liquidity pools. Indeed, they pass from an average of 17.9 sniper bots active daily to 81.3 on Ethereum, while from a few dozen to 351.7 on BSC. However, towards the end of this phase, we record a decrease in the daily active sniper bots on Ethereum, while on BSC, they remain relatively stable.

Phase 3: The All Time High and the market depression. During this time frame (November 2021 - June 2022), we have two notable events; in November 2021, the coins of both blockchains reached their maximum historical price, with Ethereum touching $4,000 dollars and BNB $650. In this circumstance, we observe a sudden increase of daily active sniper bots on Ethereum, with 249.9 daily sniper bots on average, while a bit of inflection toward down on BSC (avg. 249.9 sniper bots). Nevertheless, on both blockchains, we record in this period the most intense activity of the sniper bots (see Fig. 7 (b) and (c)), targeting 71.7% of the newly created liquidity pools on Ethereum and 70.4% on BSC. In the following months, the cryptocurrency market fell, with a drastic drop in the value of the coins. Surprisingly, in these months, the number of active daily sniper bots on BSC remains pretty steady (avg. 342.4 sniper bots), but the percentage of the sniped liquidity pools drops to 25.3%. Instead, on Ethereum, the average daily active sniper bots also decreased alongside the price drop, reaching approximately 90 bots targeting 11.8% of the liquidity pools.

Phase 4: The Merge. In this last phase (July 2022- March 2023), we notice a huge increase of daily active sniper bots on Ethereum, reaching an average number of 547.6 between October and March 2023. Meanwhile, on BSC, the number of active sniper bots remains similar to the previous phase, with 379.2 daily active sniper bots. The increase in the number of sniper bots also reflects an increase of targeted liquidity pools on Ethereum, attesting around 44%. In contrast, on BSC, the number of sniped liquidity pools decreased to 16.1%. Although we have no clear evidence about the reasons that led to this significant increase in activity on Ethereum. We can argue that Ethereum became more appealing because of its historical upgrade *The Merge* [21] (September 15, 2023), which changed the consensus protocol of the blockchain from Proof of Work to Proof Of Stake. Another factor that could have contributed to the increase of sniper bots on Ethereum is the substantial drops in the gas price. Indeed, in this last phase, we record an average gas price of 26.9 Gwei against the 105.8 Gwei of the previously considered time frame.

6 Anti-bot mechanisms

In Sec. 3.2, we show that some sniper bots have features to avoid smart contract implementations that can hinder their efficiency. In this section, we analyze the source code of smart contracts to understand how many tokens implement mechanisms that can directly or indirectly limit the action of sniper bots.

As mentioned in Section 2.1, when a smart contract is deployed on the blockchain, the transaction contains its bytecode, not its source code. Indeed, the blockchain does not store the source code of smart contracts; instead, it contains the compiled bytecode, that is not directly understandable by humans. This makes it hard for people to investigate if a smart contract is legitimate. A possible way for developers to avoid this problem is to publish the source code of their contracts. However, the only way to be sure that the published source code corresponds to the bytecode stored on the blockchain is to compile it and match it. Since this operation can be tedious, Etherscan, one of the most popular Ethereum explorers, provides a mechanism called *contract verification*. A developer can publish its Solidity source code on the Etherscan website. Etherscan automatically compiles the code and checks if the generated bytecode matches the bytecode stored in the blockchain. If there is a match, the contract becomes *verified* and is published on the Etherscan website. The exact mechanism is also offered by the BscScan website, the main explorer for the BNB Smart

Table 5. Implementation of anti-sniper bot countermeasures.

Countermeasure	Description of the implementation
Disabled trading	This strategy involves managing the trading status for a token using a boolean variable, commonly called *tradingOpen*, that is initially set to false. Only the smart contract owner can change its status to true to enable trading. We search for token smart contracts having a method (such as *tradingStatus*, *openTrading*) to set a variable that is checked in the Transfer method and that, if set to false, does not allow the token trading.
Tax during the launch window	This solution penalizes addresses trading too fast for a human by temporarily increasing the fee to 99% for blocks close to the token launch. We search for token smart contracts defining a function (typically called *getTotalFee*) that checks whether the transaction block is greater than the block of the token launch plus a certain threshold and, if not, raises the fees.
Token amount limit	This solution restricts the number of tokens that can be purchased during the launch phase. We search for token smart contracts that, in the Transfer function, check the number of tokens to transfer and, if this is greater than a specific variable (e.g., *_maxTxAmount*), revert the transaction. Some smart contracts perform this check with a specific function like *checkTxLimit*.
Transactions number limit	Some smart contracts check the number of transactions sent by an address in a given time window, setting a cooldown that blocks further transactions for that address until it expires. We look for token smart contracts implementing in the Transfer function a check that reverts the transaction if its block timestamp is lower or equal to the cooldown timer associated with the transaction recipient (e.g., *cooldownTimer[recipient]*). Some smart contracts define a function (*buyCooldown*) to set the variable managing the cooldown and its duration.
Gas price limit	Here, the goal is to slow down bots by setting a gas price limit and blocking transactions using a gas price higher than a certain threshold. We look for token smart contract defining functions, commonly called *setPriceLimit*, *setLimitsInEffetc*, or *setProtectionSettings*, to set a gas price limit.
Sniper bot blocklist	This strategy consists of blocking all the transactions sent by addresses already known for being sniper bots. We look for token smart contracts blocking the transaction if its sender belongs to the blocklist (*isSniper*). The list is updated with sniper bots' addresses buying the token at the same block of its launch.

Chain. Thus, to build the smart contracts dataset, we query the APIs [30, 31] of the two explorers to retrieve the smart contracts source code of the tokens in the liquidity pools dataset. At the end of the process, we were able to retrieve 115,613 out of 144,145 (80.2%) verified smart contracts source codes for Ethereum and 852,849 out of 1,250,913 (68.1%) for the BSC tokens.

6.1 Smart contract analysis

Once we collected the source codes for the Ethereum and BNB Smart Chain tokens, we investigated the anti-bot techniques that token smart contracts implement to hinder the action of sniper bots. Since it is not feasible to manually analyze the code of all the retrieved smart contracts, we search on the Internet for reference implementations of anti-bot measures. In particular, we search for these implementations in sector forums (e.g., OpenZeppelin [44], Ethereum StackExchanges [53]), tools for automated token creation (e.g., Tokensbygen [57], Cointool [18]), or querying Google with keywords such as: *smart contract anti-bot measures*, *anti-bot protection*, *sniper bot countermeasures*, *token sniper bot protection*. Following our research, we find six mechanisms that can hinder the action of sniper bots and 34 reference

Table 6. Smart contracts implementing anti-bot mechanisms.

	Ethereum	BSC
Disabled trading	28,707 (24.8%)	30,756 (3.6%)
Tax during the launch window	82 (0.07%)	30,301 (3.5%)
Token amount limit	36,554 (31.6%)	291,583 (34.2%)
Transactions number limit	185 (0.1%)	33,028 (3.8%)
Gas price limit	384 (0.3%)	2,046 (0.2%)
Sniper bots blocklist	53 (0.04%)	514 (0.06%)

implementations. Next, we create a regular expression for each implementation that we can use to automatically identify similar snippets of code in our smart contracts dataset. In Tab. 5, we describe the implementations for each mechanism and how we identify the token smart contracts adopting it. Moreover, We publicly release the regular expressions we used in [3]. In the following, we briefly describe the six different mechanisms and report the number of smart contracts adopting them.

Disabled trading. This mechanism allows to enable or disable the transfer of the tokens, and hence the trading, at will. As we discuss in Sec. 3.2.3, when a liquidity pool has the trading disabled at its first blocks of life, sniper bots must implement advanced features to be successful in their operations. In our dataset, we find that the smart contracts implementing this mechanism are 28,707 (24.8%) on Ethereum, and 30,756 (3.6%) on the BNB Smart Chain. The main issue with this solution is that if the owner of the token smart contract does not send its ownership to the null address when he enables trading, he could disable it again in the future, blocking its sale. To avoid this countermeasure, most sniper bots wait for some blocks before buying the token or send transactions for small amounts to check the trading status of the token.

Tax during the launch window. With this mechanism, the smart contract imposes a high tax on each token transaction (e.g., 99%) during the launch window of the liquidity pool. Sniper bots can avoid falling prey to this mechanism using the advanced feature *Wait n-blocks* (see in Sec. 3.2.3). We identify 82 (0.07%) and 30,301 (3.5%) token smart contracts on the Ethereum and BNB Smart Chain, respectively, implementing this technique. In particular, more than 91% of these smart contracts impose the tax only for the first two blocks from the token launch, while the remaining smart contracts define a different number of blocks, either with a fixed number or through a variable. The problem with this strategy is that it can also affect human users if the temporal window with increased fees lasts too long. Moreover, this strategy could suspend the trading if the token contract has a function to define new windows that raise the fee again. The solution adopted by the sniper bots to mitigate this countermeasure is the same as the previous one.

Token amount limit. This mechanism limits the number of tokens per transaction and/or address that can be purchased during the liquidity pool's early stage. Although this solution is designed mainly to counter the crypto whales (i.e., addresses that own many tokens), it also hinders sniper bots.

Nevertheless, sniper bots can easily circumvent the countermeasure that only checks the token amount in each transaction by sending multiple smaller transactions, as seen in Sec. 3.2.3. A more effective approach involves monitoring the recipient's balance or the total number of tokens purchased by the address within a specific time window. However, in our dataset, we did not find sniper bots able to circumvent the limit per address. We find 36,554 (31.6%) on Ethereum and 291,583 (34.2%) on the BSC smart contracts implementing the limit per transaction mechanism.

Transactions number limit over time. This approach prevents multiple transfers from the same address within a specified time window. It protects against sniper bots trying to bypass the token amount limit per transaction. Indeed,

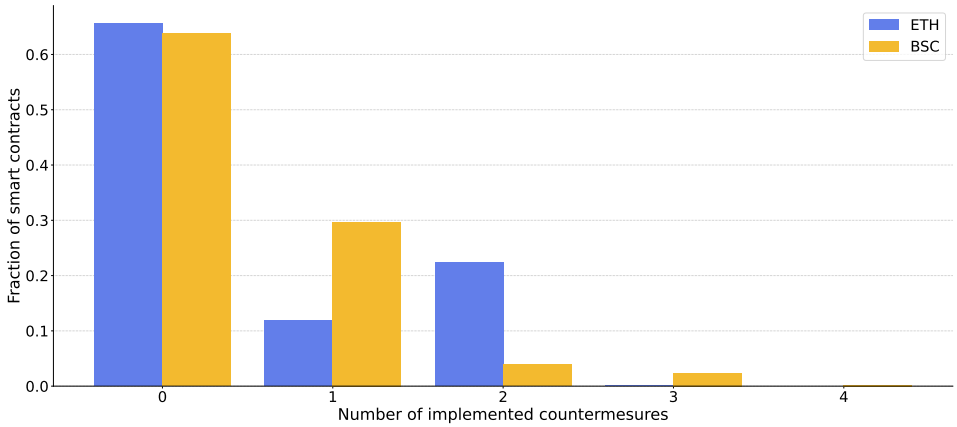

Fig. 8. Fraction of token smart contracts that implement a specific number of anti-bot mechanisms.

to circumvent this countermeasure, sniper bots could perform multiple transactions, Multiple buys (Sec. 3.2.3), where each one purchases a small amount of the target token. By restricting the number of transactions an address can make within a specific time window, this countermeasure prevents sniper bots from circumventing the token amount limit countermeasure. In particular, we identify 185 (0.1%) and 33,028 (3.8%) token smart contracts adopt this mechanism on Ethereum and BSC, respectively. This is achieved by setting for each address that has performed a transaction a cooldown that blocks further transactions for that address until it expires.

Gas price limit. As shown in Sec. 3.2.1, a common practice used by sniper bots to ensure their transactions are executed as fast as possible is to use a gas price higher than those of other transactions at that moment. Thus, a strategy to slow them down is to set a gas price limit and block transactions using a gas price higher than a certain threshold. The token smart contracts tend to set the gas price limit through a function (e.g., *setPriceLimit*). The token smart contracts implementing this strategy are 384 (0.3%) on Ethereum and 2,046 (0.2%) on the BSC. This countermeasure faces limited popularity, as an incorrect update to the gas price limit may result in service unavailability for all users during network congestion.

Sniper bots blocklist. The last mechanism consists of blocking all the transactions sent by addresses already known for using sniper bots or that perform transactions in the first blocks of the life of the liquidity pool. Overall, we find 53 (0.04%) token smart contracts on Ethereum and 514 (0.06%) on the BSC. To achieve this goal, the smart contract has a blocklist containing an initial set of addresses known for using sniper bots. Next, this list is updated with the addresses of those who try to buy in the same block of the token launch. This countermeasure can be easily circumvented by using different addresses. However, it can successfully block a sniper bot that tries to buy the token at each block starting from its launch.

Tab. 6 summarizes the number of token smart contracts implementing the analyzed mechanisms. The countermeasure limiting the token amount that can be bought is the most popular one on both blockchains (31.6% on Ethereum and 34.2% on BSC). Interestingly, we find that the second most popular mechanism to limit the sniper bot actions on BSC (*Transactions number limit*) is implemented by only 185 (0.1%) smart contracts on Ethereum. Instead, the runner-up mechanism on Ethereum (*Disable trading*) is implemented by more than 24% of the smart contracts on Ethereum and only by 3.6% on BSC.

Fig. 8 shows the fraction of smart contracts for each blockchain implementing a specific number of countermeasures. Looking at the number of mechanisms used by each token smart contract in our dataset, we find that the majority do not implement any mechanism to limit the actions of the sniper bots. Indeed, there are 13,795 (11.9%) token smart contracts on Ethereum implementing only one mechanism and 253,012 (29.6%) on BSC. 26,030 (22.5%) Ethereum token smart contracts implement more than one countermeasure. Meanwhile, only 55,756 (6.5%) token smart contracts on BSC implement two or more anti-bot mechanisms. The maximum number of mechanisms adopted is four (*disabled trading*, *tax during the launch window*, *token amount limit*, and *transactions number limit*), implemented by 1,741 token smart contracts, all running on the BSC. From our data, it appears that Ethereum and BSC token creators are equally active in contrasting the action of the sniper bots with 34.4% of the Ethereum smart contracts that implement at least a mechanism against the 36.2% on BSC.

Lastly, we analyzed smart contracts to identify those explicitly stating the use of the countermeasures we examined above to combat sniper bots. In particular, we created regular expressions—similar to the previous approach—to search for comments, variable names, or function names in the code that indicate functionalities specifically designed to hinder sniper bots. We found 1,368 smart contracts on Ethereum and 9,288 on the BSC using the *Sniper bot blocklist* countermeasure by including in the transfer function *‘require(!_isSniper[to], "ERC20: snipers can not transfer");‘*. Thus, they keep a blocklist explicitly for sniper bots and do not allow them to trade the token. Additionally, 166 contracts on Ethereum and 324 on the BSC employ the *Gas price limit* countermeasure with comments like *"// only use to prevent sniper buys in the first blocks."* We also found 78 contracts on Ethereum and 217 on the BSC using a *‘sniperProtection‘* variable to activate anti-sniper features. Moreover, 74 contracts on Ethereum and 26 on the BSC implement the *Transactions number limit over time* countermeasure with comments explicitly stating its use against sniper bots (e.g., *"// sorry about that, but sniper bots nowadays are buying multiple times, hope I have something more robust to prevent them from nuking the launch :-("*). Finally, 56 smart contracts on Ethereum and 216 on the BSC explicitly state at the beginning of their code that they include anti-sniper features.

7 Related Work

Several works study the presence of bots in the AMM market, focusing on front-running bots that perform arbitrage or sandwich attacks. Daian et al. [19] investigated the behavior of front-running bots that exploit arbitrage opportunities by monitoring the mempool. The bots scan the mempool for large buy transactions that result in an overpriced token on a particular market. They then swiftly send a transaction to purchase underpriced assets in another market and sell them in the overpriced market, capitalizing on the big buy. Qin et al. [50] propose heuristics to identify arbitrage operations and quantify their impact on the market. They found that from 2018 to 2021, arbitrage bots obtained a profit of 277.02M USD. Zhou et al. [66] studied sandwich attacks. This kind of attack is performed using two transactions. The first is placed just before the target transaction (i.e., front-run), and the second just after it (i.e., back-run). This strategy allows for profit when a significant buy is performed in the AMM. They find that on Uniswap, an attacker can obtain an average daily profit of $3,414. Instead, Torres et al. [58] study the sandwich attacks on a larger scale, taking into account several marketplaces on Ethereum, quantifying the profit obtained through sandwich attacks in 174.34M USD. Front-running bots have also been studied by Qin et al. [50], who analyze 11 million Ethereum blocks, finding more than 200 thousand attacks with an accumulated profit of $18.41M. Sniper bots have received little attention from the scientific community. They have been recently mentioned in a financial bot taxonomy proposed by [43] and partially analyzed only by Cernera et al. [13, 14]. The papers analyze blockchain data to identify rug pulls, finding 21,594 and 266,340 operations performed in the Ethereum and BSC AMM markets. Then, they identify addresses that

frequently fall prey to rug pull operations and classify them as sniper bots. With respect to their work, we perform a deep characterization of sniper bots and analysis of their implementation. Moreover, we quantify their presence outside rug pull operations and analyze their investment, gains, and success rate.

8 Discussion

How do sniping operations affect block space and transaction fees on Ethereum and BSC? To evaluate whether and how sniping operations affect the Ethereum and BSC blockchains, we estimate their impact on the process of block creation and on gas prices. As a first experiment, we examine all the blocks targeted by sniper bot operations (63,812 Ethereum blocks and 584,564 BSC blocks) and quantify the percentage of sniper bot transactions they contain. On average, sniper bot activities account for only 2.9% of the transactions in Ethereum blocks and 1.3% in BSC. However, they represent over 20% of the transactions in 910 Ethereum blocks (1.4% of the blocks where a sniper bot performs a transaction) and 1,338 BSC blocks (0.2%). We find a few cases where the sniper bot transactions represent nearly all the block transactions. For example, in Ethereum block `13,973,051` and BSC block `15,995,019`, sniper bot transactions are the 88.5% and 92.3% of the block transactions, respectively. Regarding transaction fees, our study shows a slight increase in average gas prices when sniper bots transactions are the majority in a block. Specifically, in 84.1% of Ethereum blocks and 64% of BSC blocks, where sniper bots perform more than 50% of the transactions, gas prices were higher than the daily averages by 101.5 Gwei on Ethereum and 28.9 Gwei on BSC. In conclusion, although sniper bot transactions usually are, on average, only a tiny portion of the block transactions and have minimal impact on gas prices, there are occasional instances where they almost monopolize a block, causing potential transaction delays for other users.

Do token creators welcome sniper bots? We found that many projects disapprove of sniper bot activity and have explicitly implemented mechanisms in their smart contracts to contrast them [23–25]. Moreover, popular platforms that allow users to create tokens without requiring coding expertise, such as PinkSale [47], Bitbond [8], and 20lab [1], offer the option for token creators to include anti-sniper bot features. For instance, PinkSale offers Pink Anti-Bot [46], a service to implement in token smart contracts the countermeasures described in Sec. 6.1. These include blocklisting sniper bots (*Sniper bots blocklist*), preventing trades in the first block after listing (*Disabled trading*), limiting the number of tokens per trade (*Token amount limit*), and setting a time limit on transactions (*Transactions number limit over time*). These findings suggest that token creators may disapprove of sniper bot activity. As mentioned earlier, one reason is that sniper bots increase price volatility. Additionally, sniper bots can rapidly acquire a large portion of a newly listed token's supply without protective measures. This concentration in a few wallets can damage the token's reputation and deter human investors, who may fear a sudden price crash if these wallets dump their tokens.

Are miners involved in sniper bots activities? A miner could exploit the ability to rearrange transactions within a block to conduct a sniping operation. By placing a buy transaction immediately after detecting a Mint event he would be able to purchase the token before anyone else. However, it does not guarantee profitability unless they also monitor other buy transactions within the same block and execute a sell in that block. This operation is a sandwich attack and falls outside the scope of our analysis, as detailed in Section 4.2. Alternatively, a miner could buy the token after detecting a Mint event and hold it without selling in the same block. This approach is risky, as there is no assurance that they will be able to sell the token later at a profit. For this reason, we believe miners would likely prefer more profitable MEV activities, such as rearranging transactions to ensure guaranteed profits. To corroborate this hypothesis, we conducted experiments to investigate the involvement of miners in sniper operations. Analyzing the addresses engaged in sniping operations to determine if they matched those mining the blocks, finding no overlap.

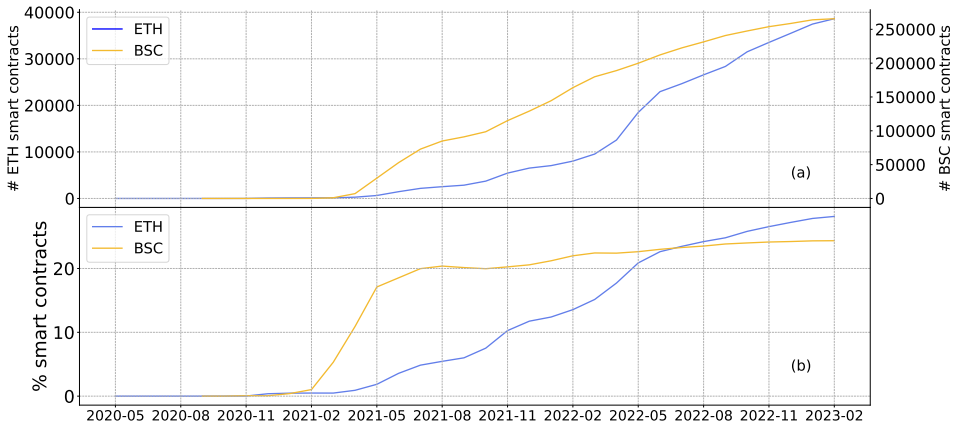

Fig. 9. Number (a) and percentage (b) of smart contracts on Ethereum (blue line) and on the BSC (orange line) implementing anti-sniper bots countermeasures over time.

Thus, based on our analysis, we believe miners prefer more lucrative MEV opportunities, rather than engaging in the risky practices associated with sniper bots.

Can we consider the sniper bots behavior a type of MEV activity? Some sniper bots analyzed in this work perform operations that could resemble the activity of MEV bots. Indeed, while MEV was initially associated with miners [19], the definition has recently expanded to include bots that exploit arbitrage opportunities, liquidations, or front-run buy orders [50, 58]. Under this broader definition, sniper bots that monitor the mempool and adjust gas fees to buy newly released tokens as quickly as possible could be classified as MEV bots. However, only about 50% of sniper bots we analyze use this approach. The others rely on monitoring smart contracts or Telegram channels, making them more similar to more traditional automated trader bots. Moreover, sniper bots have peculiarities that set them apart from other kind of MEV bots. First, the primary goal of sniper bots is to buy tokens as soon as they are released, whereas MEV bots generally seek to exploit profitable transactions regardless of when the token was launched. Additionally, sniper bots have few guarantees of profits, as they rely on token prices surpassing a set threshold, which may never occur. Finally, sniper bots typically buy a token and wait a significant amount of time before selling (on average 12 hours on BSC and 6 hours on Ethereum). In contrast, sandwich attack bots execute both buy and sell operations around single transactions, with minimal time between the two actions.

Is there an active arms race between anti-bot measures and circumvention? As discussed in Sec. 6.1, smart contracts adopt various countermeasures to combat sniper bots, which, in turn, use advanced techniques to bypass them (Sec. 3.2.3). As anti-bot features become more sophisticated, bot developers refine their algorithms to counter these defenses. An example is Maestro's Nitro [39], an advanced tool on Ethereum and BSC that allows the detection of the blocks in which a sniper bot should avoid buying tokens because of *Sniper bots blocklist* or *Tax during the launch window* countermeasures (Sec. 6.1). Then, to assess whether token creators have increased their efforts to counter sniper bot activity, we analyzed the number of smart contracts implementing anti-sniper bot measures over time (Fig. 9). Our findings show a significant shift starting in April 2021. Before this date, few smart contracts included anti-sniper bot measures. However, from April 2021 onward, there was a sharp increase in such measures, especially on the BSC, where the number of contracts with countermeasures grew from a few hundred to over 100,000 by the end of 2021.

On Ethereum, the rise was more gradual, likely due to fewer tokens and less sniper activity on that blockchain. This increase is evident also in the percentage of smart contracts that implement at least one countermeasure, which reaches nearly 12% of all token contracts on Ethereum and over 20% on BSC. While the increase on BSC has been steady, Ethereum saw a notable uptick starting in March 2022, a few months after a spike in sniper activities. By February 2023, around 40,000 (28.1% of the total) smart contracts on Ethereum and over 250,000 (24.3% of the total) on BSC had implemented at least one countermeasure. This overview shows token creators have increasingly adopted anti-sniper bot measures since March 2021, a trend that continues to grow across both blockchains.

9 Limitations

In this work, we focus only on open-source implementations of sniper bots that we find on GitHub. However, during our investigation, we also found several closed-source implementations [15] and providers that offer "Sniper bot as a service" [52]. Thus, sniper bots may offer more advanced features that we could not analyze. Our work only shows a lower bound on the usage and impact of sniper bots on the DeFi ecosystem. Finally, in our investigation of the anti-bot mechanisms implemented by smart contracts, we rely on reference implementations, which we find disclosed on the web. Even if we added some flexibility using regexes, the same techniques could have been implemented in different ways that we did not cover. Thus, in this case, our estimation of the diffusion of anti-bot mechanisms is only a lower bound.

10 Conclusion and Future Work

This paper thoroughly analyzes the phenomenon of sniper bots acting on Ethereum and BNB Smart Chain (BSC). First, we analyzed how sniper bots work, defining the phases of a sniping operation. Then, we identified sniper bots operating on AMMs of these two blockchains. We studied their behavior and quantified their economic impact on the DeFi ecosystems. We discovered that sniper bots significantly affect the price of tokens, driving up their value, in most cases, by 19.7% on Ethereum and 4.4% on BSC. Due to the low cost of each operation and the potential high gain, we revealed that even if the success rate of a sniping operation is about 50%, in the long run, their use can generate substantial earnings. Interestingly, we discovered that Rug Pull is one of the main threats for sniper bots. In this fraudulent practice, the token creators suddenly abandon it, taking with them the funds invested by users. Lastly, we described the anti-bot mechanisms implemented by smart contracts to limit sniper bots and estimated their adoption on Ethereum and BSC.

As future work, it is interesting to analyze the use of sniper bots on other blockchains, like the non-EVM-compliant ones. Finally, it is possible to explore potential techniques for mitigating sniper bots' effects and enhancing decentralized finance ecosystems' security and fairness.

Acknowledgments

This work has been partially funded by projects: MUR National Recovery and Resilience Plan, SERICS (PE00000014); and ST3P (B83C24003210001) under the "Young Researchers 2024-SoE" Program funded by the Italian Ministry of University and Research (MUR).

References

[1] 20LAB. 2024. 20LAB. https://20lab.app/.
[2] Hayden Adams, Noah Zinsmeister, and Dan Robinson. 2020. Uniswap v2 Core. (2020).
[3] Anonymous. 2023. Regex for token smart contract mechanisms that hinder sniper bots. https://doi.org/10.5281/zenodo.7604918.
[4] Dirk G Baur and Thomas Dimpfl. 2018. Asymmetric volatility in cryptocurrencies. *Economics Letters* 173 (2018), 148–151.

[5] Binance. 2023. BNB Chain Documentation. https://docs.bnbchain.world/docs/learn/intro.

[6] Binance. 2023. Proof of Authority Explained. https://academy.binance.com/en/articles/proof-of-authority-explained.

[7] Binance. 2024. Binance. https://binance.com/.

[8] Bitbond. 2024. Bitbond. https://www.bitbond.com/.

[9] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the Factors That Impact the Popularity of GitHub Repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 334–344. https://doi.org/10.1109/ICSME.2016.31

[10] Steven Brock. 2021. Scalping in Ecommerce: Ethics and Impacts. *Available at SSRN 3793357* (2021).

[11] BscScan. 2024. BscScan. https://bscscan.com/.

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

[13] Federico Cernera, Massimo La Morgia, Alessandro Mei, Alberto Maria Mongardini, and Francesco Sassi. 2023. Ready, aim, snipe! analysis of Sniper Bots and their impact on the DeFi ecosystem. In *Companion Proceedings of the ACM Web Conference 2023*. 1093–1102.

[14] Federico Cernera, Massimo La Morgia, Alessandro Mei, and Francesco Sassi. 2023. Token Spammers, Rug Pulls, and Sniper Bots: An Analysis of the Ecosystem of Tokens in Ethereum and in the Binance Smart Chain (BNB). In *32nd USENIX Security Symposium (USENIX Security 23)*. 3349–3366.

[15] cniperbot. 2023. sniperbot. https://github.com/cniperbot/sniperbot.

[16] CoinGecko. 2023. *CoinGecko*. https://www.coingecko.com

[17] CoinMarketCap. 2024. CoinMarkerCap. https://coinmarketcap.com/.

[18] CoinTool. 2023. CoinTool. https://cointool.app/createToken/bsc.

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

[20] Chris Dannen. 2017. *Introducing Ethereum and solidity*. Vol. 1. Springer.

[21] Ethereum. 2023. Paris Upgrade Specification. https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/paris.md.

[22] Etherscan. 2024. Etherscan. https://etherscan.io/.

[23] Etherscan. 2024. Meowth. https://etherscan.io/address/0x41C2f170f3F1B2c9a66edEC1e61069Fd67edd413#code.

[24] Etherscan. 2024. OdaInu. https://etherscan.io/address/0x04DC37B220A055c5F93680815F670babCD912c2C#code.

[25] Etherscan. 2024. RYU. https://etherscan.io/address/0x9a2abC0Fd363Fb4b5eAa664Da32c6DB18531190C#code.

[26] Vitalik Buterin Fabian Vogelsteller. 2015. EIP-20: Token Standard. https://eips.ethereum.org/EIPS/eip-20.

[27] Rundong Gan, Le Wang, and Xiaodong Lin. 2023. Why trick me: The honeypot traps on decentralized exchanges. In *Proceedings of the 2023 Workshop on Decentralized Finance and Security*. 17–23.

[28] Github. 2023. Github. https://github.com/.

[29] Yashu Gola. 2021. Shiba Inu could surpass Dogecoin after a 700% SHIB price rally in October. https://cointelegraph.com/news/shiba-inu-could-surpass-dogecoin-after-a-700-shib-price-rally-in-october.

[30] P.C. Kotsias. 2020. pcko1/etherscan-python. https://doi.org/10.5281/zenodo.4306855

[31] P.C. Kotsias. 2021. pcko1/bscscan-python. https://doi.org/10.5281/zenodo.4781726

[32] Massimo La Morgia, Alessandro Mei, Alberto Maria Mongardini, and Eugenio Nerio Nemmi. 2023. A game of NFTs: Characterizing NFT wash trading in the Ethereum blockchain. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 13–24.

[33] Massimo La Morgia, Alessandro Mei, Alberto Maria Mongardini, and Jie Wu. 2023. It'sa Trap! Detection and Analysis of Fake Channels on Telegram. In *2023 IEEE International Conference on Web Services (ICWS)*. IEEE, 97–104.

[34] Massimo La Morgia, Alessandro Mei, Francesco Sassi, and Julinda Stefa. 2020. Pump and dumps in the bitcoin era: Real time detection of cryptocurrency market manipulations. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 1–9.

[35] Massimo La Morgia, Alessandro Mei, Francesco Sassi, and Julinda Stefa. 2023. The doge of wall street: Analysis and detection of pump and dump cryptocurrency manipulations. *ACM Transactions on Internet Technology* 23, 1 (2023), 1–28.

[36] Defi Llama. 2024. https://defillama.com/.

[37] Defi Llama. 2024. Uniswap V2. https://defillama.com/forks/Uniswap%20V2.

[38] Defi Llama. 2024. Uniswap V3. https://defillama.com/forks/Uniswap%20V3.

[39] Maestro. 2024. Auto Snipe. https://docs.maestrobots.com/auto-snipe.

[40] Youcef Maouchi, Lanouar Charfeddine, and Ghassen El Montasser. 2022. Understanding digital bubbles amidst the COVID-19 pandemic: Evidence from DeFi and NFTs. *Finance Research Letters* 47 (2022), 102584.

[41] Bruno Mazorra, Victor Adan, and Vanesa Daza. 2022. Do not rug on me: Leveraging machine learning techniques for automated scam detection. *Mathematics* 10, 6 (2022), 949.

[42] Sarah E Michigan. 2021. Sneaker bots & Botnets: malicious digital tools that harm rather than help e-commerce. *Rutgers Bus. LJ* 17 (2021), 169.

[43] Thomas Niedermayer, Pietro Saggese, and Bernhard Haslhofer. 2024. Detecting Financial Bots on the Ethereum Blockchain. In *Companion Proceedings of the ACM Web Conference 2024* (Singapore, Singapore) *(WWW '24)*. Association for Computing Machinery, New York, NY, USA, 1742–1751. https://doi.org/10.1145/3589335.3651959

[44] OpenZeppelin. 2023. OpenZeppelin. https://forum.openzeppelin.com/.

[45] PancakeSwap. 2024. PancakeSwap. https://pancakeswap.finance/.

[46] PinkSale. 2024. Introducing Pink Anti-Bot. https://docs.pinksale.finance/pink-anti-bot/introducing.
[47] PinkSale. 2024. The Launchpad Protocol for Everyone. https://www.pinksale.finance/.
[48] Jason Carve Piper Merriam. 2023. Web3.py. https://web3py.readthedocs.io/en/stable/.
[49] Amy Cheng The Washington Post. 2021. 'Squid Game'-inspired cryptocurrency that soared by 23 million percent now worthless after apparent scam. https://www.washingtonpost.com/world/2021/11/02/squid-game-crypto-rug-pull/.
[50] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2022. Quantifying blockchain extractable value: How dark is the forest?. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 198–214.
[51] RugDoc. 2023. *RugDoc API.* https://rugdoc.io/
[52] TUF sniperbot. 2023. TUF sniperbot. https://tufsniperbot.com/.
[53] Ethereum StackExchange. 2023. Ethereum StackExchange. https://ethereum.stackexchange.com/.
[54] Dmitry Tanana. 2019. Avalanche blockchain protocol for distributed computing security. In *2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. IEEE, 1–3.
[55] Vyper Team. 2020. Vyper. https://docs.vyperlang.org/en/stable/.
[56] Telegram. 2023. *Telegram.* https://telegram.org/faq
[57] TokenByGen. 2023. TokenByGen. https://tokensbygen.com/.
[58] Christof Ferreira Torres, Ramiro Camino, et al. 2021. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *30th USENIX Security Symposium (USENIX Security 21)*. 1343–1359.
[59] Christof Ferreira Torres, Mathis Steichen, and Radu State. 2019. The art of the scam: demystifying honeypots in ethereum smart contracts. In *Proceedings of the 28th USENIX Conference on Security Symposium*. 1591–1607.
[60] Uniswap. 2022. Uniswap v2 License. https://github.com/Uniswap/v2-core/blob/master/LICENSE.
[61] Gavin Wood. 2018. Ethereum yellow paper: A formal specification of Ethereum, a programmable blockchain. 2018. *URL https://github. com/ethereum/yellowpaper* (2018).
[62] Pengcheng Xia, Haoyu Wang, Bingyu Gao, Weihang Su, Zhou Yu, Xiapu Luo, Chao Zhang, Xusheng Xiao, and Guoai Xu. 2021. Trade or trick? detecting and characterizing scam tokens on uniswap decentralized exchange. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 3 (2021), 1–26.
[63] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. 2021. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *Comput. Surveys* (2021).
[64] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. 2023. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *Comput. Surveys* 55, 11 (2023), 1–50.
[65] Dirk A Zetzsche, Douglas W Arner, and Ross P Buckley. 2020. Decentralized finance (defi). *Journal of Financial Regulation* 6 (2020), 172–203.
[66] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. 2021. High-frequency trading on decentralized on-chain exchanges. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 428–445.