Ready, Aim, Snipe! Analysis of Sniper Bots and their Impact on the DeFi Ecosystem

Federico Cernera
Sapienza University of Rome
Rome, Italy
cernera@di.uniroma1.it

Massimo La Morgia
Sapienza University of Rome
Rome, Italy
lamorgia@di.uniroma1.it

Alessandro Mei
Sapienza University of Rome
Rome, Italy
mei@di.uniroma1.it

Alberto Maria Mongardini
Sapienza University of Rome
Rome, Italy
mongardini@di.uniroma1.it

Francesco Sassi
Sapienza University of Rome
Rome, Italy
sassi@di.uniroma1.it

ABSTRACT

In the world of cryptocurrencies, public listing of a new token often generates significant hype, in many cases causing its price to skyrocket in a few seconds. In this scenario, 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 addresses that serially take advantage of sniper bots. Our findings reveal 14,029 sniping operations on Ethereum and 1,395,042 in BSC that bought tokens for a total of $10,144,808 dollars and $18,720,447, respectively. We find that Ethereum operations have a higher success rate but require a larger investment. Finally, we analyze token smart contracts to identify mechanisms that can hinder sniper bots.

CCS CONCEPTS

• **Applied computing** → **Economics**; • **Security and privacy** → **Social aspects of security and privacy**.

KEYWORDS

Trading bots, AMM, Ethereum, BNB Smart Chain

ACM Reference Format:
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 (WWW '23 Companion), April 30-May 4, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3543873.3587612

WWW '23 Companion, April 30-May 4, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9419-2/23/04…$15.00
https://doi.org/10.1145/3543873.3587612

1 INTRODUCTION

The cryptocurrency market is renowned for its high volatility [21], with cycles where the value of tokens can skyrocket and plummet [17]. This behavior is prevalent among tokens with small market capitalization, especially those that are newly listed. The rapid increase in value often triggers FOMO [3] (fear of missing out) among investors, who often purchase tokens based on hype rather than their intrinsic value. With the emergence of DeFi (Decentralized Finance) and specifically Automated Market Makers (AMMs) [40]—platforms where trading is powered by smart contracts—every blockchain user can list and make their tokens tradable. As a result, hundreds of tokens are listed on AMMs daily [39], and finding the next token worth investing in can be a challenging task. 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 bots implementations are more sophisticated than we might expect. Indeed, some of them offer features such as protection against fraudulent liquidity pools (*e.g.,* honeypots and rug pulls), as well as anti-bots mechanisms that are 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. We discover that the sniper bots phenomenon is more widespread on BSC than in Ethereum. However, after analysis of the operations conducted by these addresses, we surprisingly find that Ethereum operations have a higher likelihood of being closed with a profit, despite requiring a larger investment. Finally, we leverage Etherscan and BSCScan, two popular explorers for Ethereum and BSC, respectively, to download the source code of the smart contracts of the tokens contained in the liquidity pools dataset. We search among the retrieved smart contracts implementation of anti-bot mechanisms that can limit the action of the sniper bots. In line with our previous findings, we discover that developers of BSC token smart contracts are more active in countering bots' activities, implementing 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 choice of the target liquidity pool to the sale of the token. For each phase, we report in detail the different techniques implemented by the most popular open-source sniper bots on GitHub.
- **The impact of sniper bots**: We propose an identification methodology for addresses that take advantage of sniper bots serially. We find 161 addresses on Ethereum 819 addresses in BSC. Analyzing the operations of the identified sniper bots, we note that sniper bot users behave differently accordingly to the platform they operate. We analyze their operations to estimate their success rate and their profit. We quantify their impact on the ecosystem of liquidity pools, finding that they move a volume of 11,360.7 ETH on Ethereum and 45,606.3 BNB in BSC.
- **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 600,000 smart contracts. We observe that 17.9% token smart contracts on Ethereum and 37.36% on BSC implement at least one mechanism to hinder the action of bots.

2 BACKGROUND

2.1 Ethereum and the Binance Smart Chain

Ethereum [8] is a proof-of-stake blockchain. Its native coin, Ether (ETH), is the second cryptocurrency by market capitalization, with over 200 billion USD. Thanks to the Ethereum Virtual Machine (EVM) it is possible to execute custom machine code with smart contracts. Smart contracts are programs stored on the Ethereum blockchain, written in a high-level programming language (*e.g.,* Solidity). Smart contracts enable the creation of several kinds of assets, like, for example, fungible tokens. In Ethereum, the ERC-20 standard defines the core methods and Events that tokens should implement. The methods provide basic functionalities such as transferring tokens from one account to another or obtaining an account's current token balance. Instead, the Events are a mechanism provided by Ethereum to ease the tracking of the internal state of smart contracts. Indeed, each time the state of the smart contract change, it can trigger an Event that is written in an Event Log that developers can easily track. In the case of ERC-20 tokens, Events are used to notify significant changes, like the transfer of tokens from one account to another.

2.1.1 The Binance Smart Chain. The Binance Chain [4] was established in 2019 by Binance, one of the largest cryptocurrency exchanges. 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) [5] consensus, and its native coin, the BNB, is the third-largest cryptocurrency by market capitalization, with a value of over $48 billion. 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. Because of this compatibility, the BSC proposes itself as a faster and less expensive alternative to Ethereum.

2.2 DEX, AMM and Uniswap

Decentralized exchanges (DEXs) are cryptocurrency exchanges that allow the trade of cryptocurrencies without the need for an intermediary. The most popular DEXs leverage the Automated Market Maker model (AMM). In this model, trade matching is performed using liquidity pools, and the price of assets is fixed using a mathematical formula. Uniswap [1] is one the first dApp to use the AMM model successfully, with over 3 billion USD locked into smart contracts[1]. Fig. 1 shows, at a high level, how liquidity pools work in Uniswap. A *liquidity pool* is a smart contract that contains a pair of ERC-20 tokens (A, B) that users can swap. Users that want to invest in the liquidity pool provide both tokens (A, B) to the smart contract, becoming *liquidity providers*. To keep track of the share of the liquidity owned by each investor, liquidity pools 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 contract. 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.

Uniswap implements the AMM model using mainly three smart contracts.

- The *Factory* contract is used to create the smart contract that handles liquidity pools. It is responsible for creating one and only one liquidity pool for each token pair. Each time a new liquidity pool is created, 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 changes in the status of the liquidity pool. 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 pool created by the Factory smart contract implements these Events.
- The *Router* offers an entry point to interact with the liquidity pools. Interacting with the Router, it is possible to create liquidity pools, add or remove liquidity, and swap tokens.

3 SNIPER BOTS

Sniper bots are software applications that monitor a specific activity to automatically perform an action before anyone else [9]. Examples of these bots are "Scalping bots," [7] programs designed to purchase limited-availability goods quickly. These kinds of bots have been used to buy limited-edition sneakers [24] and Nvidia GPUs during the 2021 graphic card shortage [7]. The goal of bots' users is usually to resell the purchased items at a higher price. In the blockchain world, sniper bots are typically used to buy tokens as soon as they are listed on an AMM platform.

[1] Data retrieved from DefiLlama [22], a popular DeFi statistics aggregator, in 2023-01-10

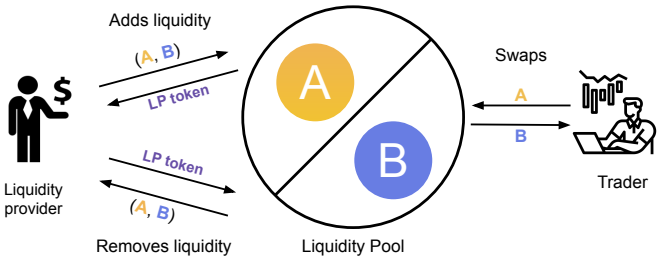

Figure 1: Liquidity pool and its main operations.

3.1 Sniper bots dataset

To understand how sniper bots are implemented and the features they offer, we leverage Github [16], one of the most popular Internet hosting services for software. We systematically search sniper bots on Github using keywords such as: "Sniper bots," "Sniping bots," and other similar variations. This research yielded hundreds of open-source repositories that are impractical to analyze manually. Therefore, we decide to focus on sniper bots that have some popularity. To do so, we leverage GitHub's star ranking system [6] as a metric to infer popular repositories. We decided to analyze only sniper bots with at least 15 stars. Using this criterion, we select 70 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) and 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 to use the bots. In the end, considering only the open-source implementations, we focus on analyzing 28 repositories.

3.2 The anatomy of sniper bots

Analyzing the sniper bots' source codes, 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 [29] blockchain. In particular, 17 sniper bots exclusively support the BSC, three support Ethereum, and seven offer multi-chain support, being able to target both BSC and Ethereum. Looking more in detail at the implementations, 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-targets 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 using these bots is to buy as many different tokens as possible, hoping that at least a few of them will gain value in the future. In our dataset, we find 3 implementations of multi-targets sniper bots.

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 kinds of bots aim to buy the token as soon as it is listed, the sniper bot looks for newly created liquidity pools that are 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 that 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.** As we will discuss in §5, some token smart contracts implement techniques to avoid bot interactions. Thus, the sniper bot can implement workarounds to evade detection and still buy the token. This is an advanced feature that we find implemented in 10 cases. 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.
(5) **Sell the token.** The sniper bot can also offer the possibility to sell 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 the liquidity pool containing the target token as soon as it is available is to leverage the blockchain mempool – the list of pending transactions waiting to be included in 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 if the bytecode of the transaction contains the signature of the *addLiquidity* function of the Uniswap router (*i.e.,* the function that is used to add liquidity to a liquidity pool).

Leverage Uniswap smart contracts. These kind of sniper bots directly interacts with the smart contracts of the AMM. In particular, it calls the *getPair* function of Uniswap's Factory contract at regular time intervals. This function takes a pair of token addresses as input and returns as output the address of the liquidity pool that contains the pair, if it exists, or the zero address if it does not. Thus, the

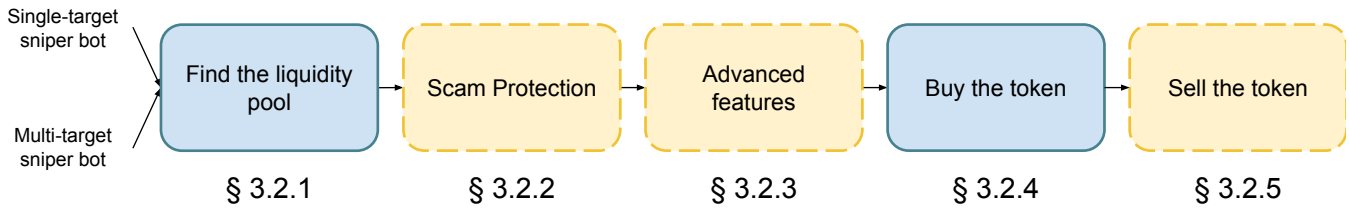

Figure 2: The phases of a sniper bot's execution. We report in blue the phases we always find implemented by sniper bots. Instead, we report in yellow the optional phases that a sniper bot can implement to improve its usability.

sniper bot can use *getPair* providing as input the token to snipe and the valuable token they want to buy the token with (*e.g.,* ETH). This method is slower than the previous one. Indeed, in order for the Uniswap smart contract to be updated, the transaction that updates its status must be confirmed in the blockchain.

However, even if the liquidity pool exists, there is no guarantee that it contains liquidity. Indeed, a user can create a liquidity pool but not add tokens to it, making any kind of swap impossible. Thus, once identified the liquidity pool, 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. When this quantity becomes different than zero, the liquidity has been added, and the sniper bot can perform the swap. 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 § 2.2, this Event is emitted by the Factory contract of Uniswap each time a new liquidity pool is created. 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 actually 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 [35], a very popular messaging app with more than 700 million active users, as a source to discover new liquidity pools. Indeed, on Telegram, there are many channels—public groups where only the admin can write [20]—dedicated to token release announcements. These sniper bots use Telegram APIs to monitor a list of channels. The sniper bot parses newly-published messages of these channels, looking for the address of a liquidity pool created on the target AMM. For instance, we find a sniper bot that monitors Telegram channels [33, 34] reporting newly-listed tokens by the CoinMarketCap [12] and CoinGecko [11], 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. 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. We find that sniper bots often perform checks to avoid buying scams or suspicious tokens. This is not surprising, as anecdotal evidence (*e.g.,* SquidGame [27]) and previous works [9, 23, 39], have shown that investing in liquidity pools can be risky as thousand of tokens are purposely created to perform scams. The sniper bots' countermeasures are mainly designed to prevent two threats: rug pulls [9, 39] and honeypot tokens [38]. 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. The goal of this practice is 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 API of RugDoc [30], 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 level of risk of the target token. If the estimated level of risk is acceptable, the sniper bot will proceed with buying 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 (for Ethereum) or BSCScan (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.

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

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. 5, several token smart contracts implement techniques to hinder sniper bots or bots in general (*e.g.,* trading bots).

Wait *n*-blocks. This feature enables the user to specify the number of blocks the sniper bot waits to purchase after the liquidity is added. This precaution is 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 buy the token too quickly, prohibiting subsequent token transfers from the blocklisted addresses. Others

may impose a very high fee on purchase transactions (*e.g.,* 99% of the acquired token returns to the liquidity pool) executed on the first blocks the liquidity is added.

Check trading enabled. Some token smart contracts implement the possibility to enable and disable the transfer of the token at will. The token creator can use this functionality for different technical or marketing reasons. To handle this case, some sniper bots implement a procedure to infer when a token enables the transfer functionality as soon as possible. The sniper bot sends a small transaction. If the transaction succeeds, the bot performs a second transaction and buys the intended amount of tokens. Otherwise, we find two different approaches implemented by the sniper bots in our dataset: In the first, the sniper bot starts to poll the liquidity pool's smart contract monitoring the token's price. If the price oscillates, the sniper bot infers that the transfer is enabled and attempts to buy the token. Instead, with the second approach, the sniper bot monitors the mempool looking for a transaction that contains the bytecode of commonly known functions used to enable the transfer of the token, such as: *openTrade*, *enableTrading*, *tradingStatus*.

Multiple buys. There are smart contracts that restrict the number of tokens an address can buy in the same transaction. This feature 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 offer the possibility to buy the desired amount of tokens using multiple buy transactions, working around the smart contract limitation.

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.

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. Indeed, the 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 the sniper bots provide an automatic way to buy tokens, not all of them offer the feature to sell them automatically. Indeed, we find that 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 called the Trailing Stop. With the Trailing Stop, the sniper bot continuously tracks the token's price. If the maximum value of the token falls below a given percentage, the sniper bot automatically sells the token.

4 SNIPER BOTS DETECTION

In the previous section, we focused on 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.

4.1 Liquidity pools dataset

To study the sniper bots, we create the liquidity pools dataset, a collection of liquidity pools and their operations in Ethereum and BSC. To retrieve the data, we run an Ethereum and a BNB Smart Chain node on our machine and synchronize the two blockchains. Then, we use Web3 [26], a Python library that allows interaction with EVM-compliant nodes to query the blockchains and obtain the data from their inception to March 2022. To collect the data, we use the same approach of previous works [9, 23, 39]. In particular, we parse the Event Logs of both blockchains, collecting Events compliant 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, their block of creation, the transaction hash, and the address that created the pool. We find 70,656 liquidity pools on Ethereum and 972,467 on BSC, which contain in their pairs 61,507 unique tokens in Ethereum and 840,862 unique tokens in BSC.
- **Mint:** By collecting Mint events, we infer when liquidity providers added liquidity to the pool. From the Event, 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 2,359,333 Mint events in Ethereum and 26,972,440 Mint Event in 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 82,430,138 Swap events in ETH and 749,188,792 Swap events in BSC.

4.2 Sniper bots identification

As a first step towards understanding how sniper bots are operatively used, we have to identify them. Although sniper bots can target any liquidity pool pair, we focus on sniper bots that target liquidity pools containing the native coin of the blockchain (BNB or ETH), which are 86.5% and 91.3% of the liquidity pools on Ethereum and BSC, respectively. Narrowing our research on these liquidity pools allows us to easily 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. Conversely, we define as a sell operation any swap that takes an ERC-20 (BEP-20) token as input and provides as output ETH (BNB). Furthermore, considering the speculative nature of sniper

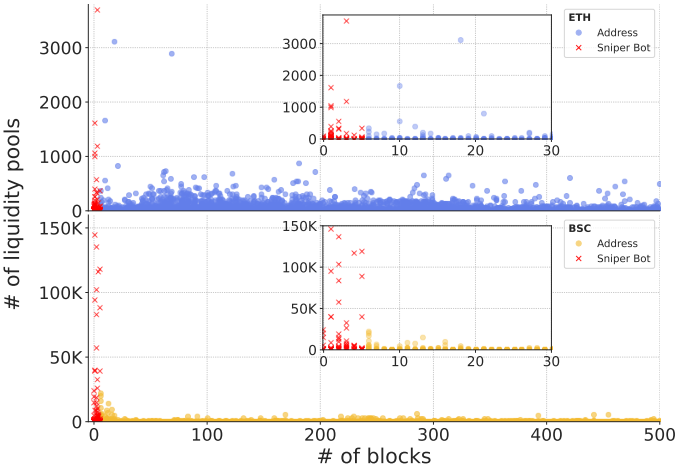

Figure 3: The scatter plot shows on the y-axis the number of liquidity pools where each address performed buy operations. On the x-axis, we plot the blocks elapsed between the buy operations and the first time liquidity is added to the pool.

bots, it is reasonable to assume that a user would never snipe a liquidity pool he created. Thus, we remove from our dataset all the buy and sell operations performed in the liquidity pool created by the same address performing the swap (3,201,920 swaps).

As we saw in the previous subsections, sniper bots are developed to perform buy operations immediately after the liquidity is added to the liquidity pool. However, in some cases, they can not always buy the token in the same block the liquidity is added, but they have to wait for some blocks to be sure they do not fall prey to scams or high taxes (see Sec.3.2.3). Even if it is difficult, a standard user could swap into a new liquidity pool a few blocks after it has been created. Thus, to avoid this case, we focus only on addresses that serially take advantage of sniper bots. Moreover, in our identification process, we have to consider that the user can operate with the same address for sniping tokens but also for his regular trading activities. Thus, some sniper bots' addresses could have operations carried out far from the creation of the liquidity pool. With these considerations, we outline two conservative thresholds to identify sniper bots' addresses by looking at their activities:

- ❖ At least 90% of the address buy operations have to be performed into 5 blocks from the block in which the liquidity was added for the first time to the liquidity pool.
- ❖ The address has to perform a buy operation in at least 5 different liquidity pools.

In the following subsection, we analyze the addresses selected by applying these two thresholds.

4.3 Results

Fig. 3 shows a scatterplot where each address is represented by a dot (blue for Ethereum addresses and yellow for BSC addresses). The *y-axis* displays the number of liquidity pools the address has traded in, and the *x-axis* shows the 90th percentile of the time intervals in blocks between the first addition of liquidity to the pool and the address's buy operations. Both figures contain a zoom of the first 30 blocks. We leverage the Mint events in our liquidity pools dataset to calculate the time elapsed from the buy operation and when the liquidity is added for the first time. We indicate with red crosses the addresses selected using our thresholds. As we can see, these addresses perform buy operations extremely close to the first liquidity addition and in hundreds of liquidity pools, exhibiting a pattern highly compatible with sniper bots' operations. For the remained sections, we will refer to these addresses as "sniper bots". Analyzing them, we discover that:

Table 1: Summary of sniper bots operations and their profits.

Metric	Ethereum	BSC
# Liquidity pools	55,678	710,515
# Sniper bots	161	819
# Operations	14,029	1,395,042
Avg. buy	0.75 ETH	0.03 BNB
Avg. gain	0.84 ETH	0.08 BNB
Success rate	25.6%	7.0%

Sniper bots are more prevalent in the BSC. Using our thresholds, we select 161 addresses on Ethereum, and 819 addresses on BSC, performing 15,052 buy operations and 1,440,945 operations, respectively. The total Ethereum and BSC liquidity pools targeted are, respectively, 7,879 and 198,786. To confirm that these addresses are sniper bots, we quantify how many performed a buy operation in the same block where the liquidity is added to the pool for the first time. This operation is virtually impossible to perform by a human, as it requires monitoring the mempool. We find that 144 (89.4%) addresses on Ethereum and 512 (62.5%) on BSC perform at least a buy operation at the same block of the first liquidity addition.

Sniper bots use different strategies in Ethereum and BSC. Ethereum sniper bots perform, on average, fewer operations than BSC sniper bots (93 vs. 1,759). However, they tend to invest higher sums than BSC sniper bots, with an average of 0.75 ETH ($673) against 0.03 BNB ($13). These different behaviors are arguably dictated by the different costs of fees on the two blockchains. Indeed, computing the fee spent to buy tokens by snipers bots, we find that, on average, they spent 0.019 ETH ($23.1) on Ethereum while 0.001 BNB ($0.46) on BSC.

Sniper bots have a relevant economic impact. Summing up the buy operations, we observe that sniper bots have a significant economic impact. These bots invest 11,360.7 ETH ($10,144,808) in Ethereum and 45,606.3 BNB ($18,720,447) in the BSC.

4.4 Gains

In this section, we analyze in detail the operations performed by sniper bots to estimate their profitability. For each sniper bot, we aggregate all the buy and sell operations performed on a token in a single *sniping operations*. Indeed, as mentioned in Sec. 3.2.3, sniper bots can buy or sell a token using multiple transactions. After this aggregation, we find 14,029 sniping operations in Ethereum and 1,395,042 sniping operations in BSC. For each sniping operation, we estimate its profit using the following formula:

$$balance = T_{out} - T_{in} - fees \quad (1)$$

Where T_{out} is the profit obtained by the sell operations, T_{in} is the amount spent to buy the token, and $fees$ is the transaction fees paid for buy and sell operations. In the following, we divide the operations into successful and unsuccessful, considering an operation successful if the *balance* is strictly positive.

Successful operations. Interestingly, we find that in BSC only 96,809 (7.0%) of the sniping operations are successful. The success rate is better on Ethereum, with 3,571 operations (25.6%). Moreover, we find that the average gain of Ethereum (0.84 ETH) is higher than the average BSC gains (0.08 BNB). Even if sniping operations are unsuccessful on average, we find some extreme cases of profit indicating that sniping tokens can be a high-risk, high-reward strategy. In particular, we find an address[2] that performs a sniping operation with a profit of 299.8 ETH. The address buys 1.86M TrustSwap [13] tokens paying 90 ETH (0.00004 ETH for each token), exactly in the same block when the liquidity is added to its liquidity pool (block 10426750). The sniper bot sells 1M of TrustSwap tokens 23 blocks after the buy, with a price increase of 600% (0.00024 ETH). Then it sells the remaining tokens for a similar price in 3 subsequent transactions for a total of 390 ETH. If we subtract the initial investment of 90 ETH and the transaction fees, the address profits 299.8 ETH from the operation.

Unsuccessful operations. Most of the sniping operations are unsuccessful. Indeed, 10,458 (74.5%) Ethereum operations and 1,298,233 (93.0%) BSC operations are unsuccessful. We notice that almost all BSC operations (85.5%) and a large fraction (48.8%) of Ethereum sniping operations are unsuccessful because the sniper bots did not sell the token. Possibly, these addresses did not sell the token because they could not do so. Indeed, Cernera et al. [9] show that almost 60% of BSC and Ethereum liquidity pools have a rug pull in the first day of their life. Thus, it is possible that the sniper bots did not sell the tokens before all the liquidity was removed from the pool. In the cases where the sniper bots sell the tokens, the loss is generally not too high, with 0.11 ETH ($108) in Ethereum and 0.01 BNB ($4.1) in BSC. Tab. 1 resumes our findings about sniper bots and their profits.

5 ANTI-BOT MECHANISMS

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 Sec. 3.2.2, Etherscan and BSCScan offer the possibility to upload on their website the source code of a smart contract to verify it. Thus, to build the smart contracts dataset, we query the APIs [18, 19] of the two explorers to retrieve the smart contracts source code of the tokens contained in the liquidity pools dataset. At the end of the process, we are able to retrieve 47,619 out of 61,507 (77.42%) verified smart contracts source codes for Ethereum and 545,048 out of 840,862 (64.82%) for the BSC tokens.

5.1 Smart contract analysis

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 [25], Ethereum StackExchanges [32]), tools for automated token creation (*e.g.,* Tokensbygen [36], Cointool [14]), 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 different mechanisms that can hinder the action of sniper bots and 34 reference 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. 3 in the Appendix, 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 [2].

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 4,584 (9.62%) on Ethereum, and 15,170 (2.78%) on the BNB Smart Chain.

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 9 (0.018%) and 15,540 (2.85%) token smart contracts on the Ethereum and BNB Smart Chain, respectively, implementing this technique. In particular, more than 88% 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.

Token amount limit. This mechanism consists in limiting the number of tokens per transaction and/or per address that can be purchased during the early stage of the liquidity pool. Although we find sniper bots successfully bypassing the transaction limit (Sec. 3.2.3), we have no evidence of sniper bots being able to evade the limit per address. We find 7,749 (16.27%) on Ethereum and 189,465 (34.76%) on the BSC smart contracts implementing the limit per transaction mechanism. In contrast, only 18 on Ethereum and 24,714 on the BSC implement the limit per transaction.

Transactions number limit over time. To solve the problem of multiple transactions used to circumvent the previous mechanism, some smart contracts do not permit multiple transfer operations requested by the same address in a given time window. In particular, we identify 10 (0.02%) and 13,018 (2.38%) token smart contracts adopt this mechanism on Ethereum and BSC, respectively.

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. Using this approach, we find the token smart contracts implementing this strategy are 143 (0.3%) on Ethereum and 1,157 (0.21%) on the BSC.

Sniper bots blocklist. The last mechanism consists in blocking all the transactions sent by addresses already known for being sniper

[2] 0xc0c5c6ea185b331ffc97499fb6bf7c1f1a0fc48c

Table 2: Smart contracts implementing anti-bot mechanisms.

	BSC	Ethereum
Disabled trading	15,170 (2.78%)	4,584 (9.62%)
Tax during the launch window	15,540 (2.85%)	9 (0.018%)
Token amount limit	189,465 (34.76%)	7,749 (16.27%)
Transactions number limit	13,018 (2.38%)	10 (0.02%)
Gas price limit	1,157 (0.21%)	143 (0.3%)
Sniper bots blocklist	464 (0.08%)	75 (0.15%)

bots or that perform transactions in the first blocks of life of the liquidity pool. Overall, we find 464 (0.08%) token smart contracts on the BSC and 75 (0.15%) on Ethereum.

Tab. 2 summarizes the number of token smart contracts implementing the different mechanisms analyzed. As we can see, the strategy that limits the token amount that can be bought is the most popular one on both blockchains (16.27% on Ethereum and 34.76% on BSC). Interestingly, we find that the second most popular mechanism to limit the sniper bot actions on BSC (*Increased fees*) is implemented by only nine (less than 0.02%) smart contracts on Ethereum. Instead, the runner-up mechanism on Ethereum (*Disable trading*) is implemented by more than 9% of the smart contracts on Ethereum and only by 2.78% on BSC.

Looking at the number of mechanisms used by each token smart contract in our dataset, we find that usually, they do not implement any mechanism to limit the actions of the sniper bots. Indeed, there are 9% token smart contracts on Ethereum and 31.4% on BSC implementing only one mechanism and very few more than one. The maximum number of mechanisms adopted is four (*disabled trading*, *token amount limit*, *transactions number limit*, and *increased fees*), implemented by 1,024 token smart contracts, all running on the BSC. From our data, it appears that BSC token creators are more active in contrasting the action of the sniper bots with 37.36% of the smart contracts that implement at least a mechanism against the 17.9% on Ethereum. This is probably because, as we have seen in Sec. 4.3, the sniper bot phenomenon is more spread on the BSC ecosystem than on Ethereum.

6 RELATED WORK

Several works study the presence of bots in the AMM market, with a particular focus on front-running bots that perform arbitrage or sandwich attacks. Daian et al. [15] 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 on another market and sell them on the overpriced market, capitalizing on the big buy. Qin et al. [28] propose heuristics to identify arbitrage operations and quantify their impact on the market. They find that from 2018 to 2021 arbitrage bots obtained a profit of 277.02M USD. Zhou et al. [41] 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 making a 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, Qin et al. [28] 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 Torres et al. [37], they 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 since they have been partially analyzed only by Cernera et al. [9]. The paper analyzes blockchain data to identify rug pulls, finding 21,594 and 266,340 operations performed respectively in the AMM markets of Ethereum and the BSC. 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.

7 LIMITATIONS

In this work, we focus only on open-source implementations of sniper bots that we find on GitHub. However, during our investigation, we find also several closed-source implementations [10] and providers that offer "Sniper bot as a service" [31]. Thus, there may be sniper bots that offer more advanced features that we could not analyze. From the point of view of the sniper bots identification, we purposely focus on detecting addresses that perform sniping operations serially. However, it is also possible that some addresses use single-target sniper bots to perform only one operation or rotate the addresses they use. For these reasons, 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, also in this case, our estimation of the diffusion of anti-bot mechanisms is only a lower bound.

8 CONCLUSION AND FUTURE WORK

This paper provides a thorough analysis of the phenomenon of sniper bots operating on Ethereum and BSC. First, we analyzed how sniper bots work, defining the phases composing a sniping operation. Then, we identified sniper bots operating on AMMs compatible with Uniswap and its forks. We studied their behavior and quantified their economic impact on the DeFi ecosystems. 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 investigate the reasons for the low success rate of sniper bots, especially on BSC. Another possible direction is to assess the impact of sniper bots on the listing price of the target token. Finally, extending our analysis to addresses that do not use sniper bots serially would be valuable for a more comprehensive understanding of the phenomenon.

ACKNOWLEDGMENTS

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. This work was partially supported by the MIUR under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science of Sapienza University.

REFERENCES

[1] Hayden Adams, Noah Zinsmeister, and Dan Robinson. 2020. Uniswap v2 Core. (2020).
[2] Anonymous. 2023. Regex for token smart contract mechanisms that hinder sniper bots. https://doi.org/10.5281/zenodo.7604918.
[3] Dirk G Baur and Thomas Dimpfl. 2018. Asymmetric volatility in cryptocurrencies. *Economics Letters* 173 (2018), 148–151.
[4] Binance. 2023. BNB Chain Documentation. https://docs.bnbchain.world/docs/learn/intro.
[5] Binance. 2023. Proof of Authority Explained. https://academy.binance.com/en/articles/proof-of-authority-explained.
[6] 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
[7] Steven Brock. 2021. Scalping in Ecommerce: Ethics and Impacts. *Available at SSRN 3793357* (2021).
[8] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3, 37 (2014).
[9] Federico Cernera, Massimo La Morgia, Alessandro Mei, and Francesco Sassi. 2023. Token Spammers, Rug Pulls, and SniperBots: An Analysis of the Ecosystem of Tokens in Ethereum and the Binance Smart Chain (BNB). In *32th USENIX Security Symposium (USENIX Security 23)*.
[10] cniperbot. 2023. sniperbot. https://github.com/cniperbot/sniperbot.
[11] CoinGecko. 2023. *CoinGecko*. https://www.coingecko.com
[12] CoinMarketCap. 2023. *CoinMarketCap*. https://coinmarketcap.com/
[13] Coinmarketcap. 2023. *TrustSwap*. https://coinmarketcap.com/currencies/trustswap/
[14] CoinTool. 2023. CoinTool. https://cointool.app/createToken/bsc.
[15] 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.
[16] Github. 2023. Github. https://github.com/.
[17] 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.
[18] P.C. Kotsias. 2020. pcko1/etherscan-python. https://doi.org/10.5281/zenodo.4306855
[19] P.C. Kotsias. 2021. pcko1/bscscan-python. https://doi.org/10.5281/zenodo.4781726
[20] Massimo La Morgia, Alessandro Mei, Alberto Maria Mongardini, and Jie Wu. 2021. Uncovering the Dark Side of Telegram: Fakes, Clones, Scams, and Conspiracy Movements. *arXiv preprint arXiv:2111.13530* (2021).
[21] 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.
[22] Defi Llama. 2023. Defi Llama. https://defillama.com/.
[23] 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.
[24] Sarah E Michigan. 2021. Sneaker bots & Botnets: malicious digital tools that harm rather than help e-commerce. *Rutgers Bus. LJ* 17 (2021), 169.
[25] OpenZeppelin. 2023. OpenZeppelin. https://forum.openzeppelin.com/.
[26] Jason Carve Piper Merriam. 2023. Web3.py. https://web3py.readthedocs.io/en/stable/.
[27] 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/.
[28] 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.
[29] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. 2019. Scalable and probabilistic leaderless BFT consensus through metastability. *arXiv preprint arXiv:1906.08936* (2019).
[30] RugDoc. 2023. *RugDoc API*. https://rugdoc.io/
[31] TUF sniperbot. 2023. TUF sniperbot. https://tufsniperbot.com/.
[32] Ethereum StackExchange. 2023. Ethereum StackExchange. https://ethereum.stackexchange.com/.
[33] Telegram. 2023. *CoinGecko & CoinMarketCap Listing Alerts Premium*. https://t.me/CMC_CG_listing_alerts
[34] Telegram. 2023. *Coinmarketcap Fastest Alerts*. https://t.me/CMC_fastest_alerts
[35] Telegram. 2023. *Telegram*. https://telegram.org/faq
[36] TokenByGen. 2023. TokenByGen. https://tokensbygen.com/.
[37] Christof Ferreira Torres, Ramiro Camino, et al. 2021. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *30th USENIX Security Symposium (USENIX Security 21)*. 1343–1359.
[38] 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.
[39] 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.
[40] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. 2021. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *Comput. Surveys* (2021).
[41] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. 2021. High-frequency trading on decentralized on-chain exchanges. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 428–445.

A APPENDIX

Table 3: Implementation of anti-sniper bot mechanisms.

Mechanism	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 aims to penalize 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 block of the transaction 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 amount of tokens to transfer and if this is greater than a certain 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 setting a gas price limit and block 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 in 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.