

TipsyCoin Technical Whitepaper v1.0

Mittens
mittens@tipsycoin.io

Goldpepperr
goldpepperr@tipsycoin.io

March 2022

Preface

This is a ‘preprint’ of the TipsyCoin technical whitepaper. If you wish to participate in an informal and decentralized ‘peer review’ process, please email us at the above email addresses with any thoughts, questions, or suggestions.

The purpose of this technical whitepaper is to discuss in a more nuanced way ‘tax on transactions’ based tokens that rose to prominence particularly during the SafeMoon ‘craze’ of early 2021. TipsyCoin, too, draws much of its tokenomic origins from SafeMoon mechanics, but has been redesigned in a number of ways that we believe make it a step forward for ‘tax on transaction’ tokens. While we are building a revolutionary play-to-earn metaverse game, TippyVerse, we need to ensure that its primary governance token, or currency, is sustainable and efficient in its own right.

As always, you can view our website at: tipsycoin.io, our GitHub here: github.com/TipsyCoin, our CertiK audit here: www.certi.kom/projects/tipsycoin, and our audit write-up on our Medium page, here: <https://medium.com/@tipsycoin/tipsycoins-certi-k-audit-a5bf5afdc8a>.

Abstract

While ‘tax on transaction’ tokens such as the popular SafeMoon and its derivatives have been popular in the DeFi space, and do contain genuinely novel ideas – the smart contracts running these tokens have design flaws that are rarely explored from a technical perspective.

Users, generally unequipped to comprehend the nuanced criticisms of these tokens, often substitute the required technical comprehension for ad-hoc justifications based on price movement – if the line goes up, the project must be good. Compounding matters, the often unnecessary complexity of the SafeMoon contract makes it difficult to understand and therefore iterate and improve.

In this technical whitepaper, we discuss some of the issues found in SafeMoon (and derivatives) in the context of their smart contract code, the CertiK audit, and several million transactions worth of data collected on PancakeSwap (PCS). With this, we propose TipsyCoin, a redesigned ‘step forward’ in tax on transaction tokens that doesn’t suffer from the considerable issues found in SafeMoon. TipsyCoin is a simpler, more maintainable smart contract that provides greater reliability in transactions.

1. Introduction

The rise of DeFi in 2021 was meteoric. In December 2020, decentralized exchange (DEX) volume was just over 20 billion USD, yet by May 2021, it has rocketed up to over 140 billion USD on Ethereum mainnet alone ¹.

One of the most popular tokens during this DeFi hype sequence was SafeMoon, which attracted over 800 million USD in volume during the May 2021 period ². In addition to its significant trading volume during 2021, it was also notable as a token frequently discussed and endorsed on social media by a number of popular celebrities and online personalities, including: Jake Paul, Nick Carter, Soulja Boy, and Lil Yachty. Details, examples, and allegations of these celebrity endorsements can be found in the recent class action lawsuit against SafeMoon³, but examples include Jake Paul's Tweet (to his 4.2 million followers): "Everyone needs #SAFEMOON or this will be you." ⁴, followed by an image of a UFC fighter appearing bloody and beaten, or Lil Yachty's Tweet (to his 5.4 million followers) of the #SAFEMOON hashtag three times in a row ⁵.

Whilst this paper could never hope to adjudicate the pending class action lawsuit against SafeMoon, it will attempt to deliberate on some of the technical aspects that appear in the SafeMoon smart contracts, and hopefully portray our discussion in a manner that is easy to understand.

So, to recap, in this paper we discuss the SafeMoon contracts, identify some perceived issues, and show how TopsyCoin addresses these by simplifying the reflection and liquidity functions to make them more readable (maintainability) with lower complexity (gas cost) and how it utilizes a network of smart contracts / gnosis vaults to hold LP and tokens not in circulation, so they're never in the hands of an Externally Owned Account (EOA), essentially decentralizing control.

We focused on these issues, specifically, because they were the target of comment and complaint by CertiK's audit on SafeMoon ⁶, particularly the decentralisation aspect which could only be 'Partially Resolved' due to SafeMoon's contracts already being deployed and immutable.

1.1 TopsyCoin Background

TopsyCoin is a new GameFi project to be launched in 2022, containing a 'tax on transfer' primary token named TopsyCoin to launch on March 15th on the Binance Smart Chain (BSC), an ERC721 NFT collection launching in April on Ethereum Mainnet, and a Minecraft world & blockchain plugin with a dedicated in-game currency launching shortly afterwards.

While this paper will specifically focus on the transaction taxed main token, TopsyCoin, more information on the other aspects of the project may be found at our project website, <https://tipsycoin.io>.

A number of links to TopsyCoin articles, code, and more, have also been provided in the preface section at the beginning of this paper.

¹ <https://dune.xyz/queries/1847>

² <https://nomics.com/assets/safemoon-safemoon-v1-old/history>

³ <https://www.classaction.org/media/merewhuader-et-al-v-safemoon-llc-et-al.pdf>

⁴ <https://twitter.com/jakepaul/status/1375868696980123649>

⁵ <https://twitter.com/lilyachty/status/1377293537264418818>

⁶ <https://www.certi.kom/projects/safemoon>

1.2 SafeMoon Background

SafeMoon is a transaction tax token that was launched on the Binance Smart Chain (BSC) network on March 1st 2021⁷. BSC is an Ethereum Virtual Machine (EVM) compatible blockchain launched by the Binance group in April 2019⁸. EVM compatibility means that BSC is compatible with, and runs on, the same smart contract code as on the Ethereum network.

BSC has a number of minor differences when compared to mainnet, which are generally beyond the scope of this paper. However, a couple of the *key* changes are the gas fees on BSC being deliberately minimized, with a minimum of 5 Gwei per unit of gas and an average of 6.5 Gwei⁹.

Gas cost is hugely reduced compared to Ethereum, which frequently averages over 100 Gwei¹⁰ (a reminder that almost all operations on the EVM, including mathematical operations and data storage incurs a gas cost as described in the Ethereum yellow paper¹¹, and that the final USD cost of a transaction on the network is defined as $\text{ETH_Price} * \text{Gas_used} * \text{Gas_fee} / 1\text{e}^{18}$). The reduction in gas fee means that more complex code can be executed on the BSC network at a far lower USD cost. While this occasionally results in severely degraded performance of the network during heavy usage periods¹², it does allow for tokens on the BSC network to perform potentially complex operations, even on 'basic' transactions such as transfers, without incurring prohibitive costs.

The popularity of low fee token creation and trading on BSC is also immediately evident in the number of token contracts deployed to DEX's, with BSC's PancakeSwap V2 factory tracking 806,945 created coin pairs, compared to one of Eth Mainnet's premier DEX's, Uniswap V2, tracking just 64,827.

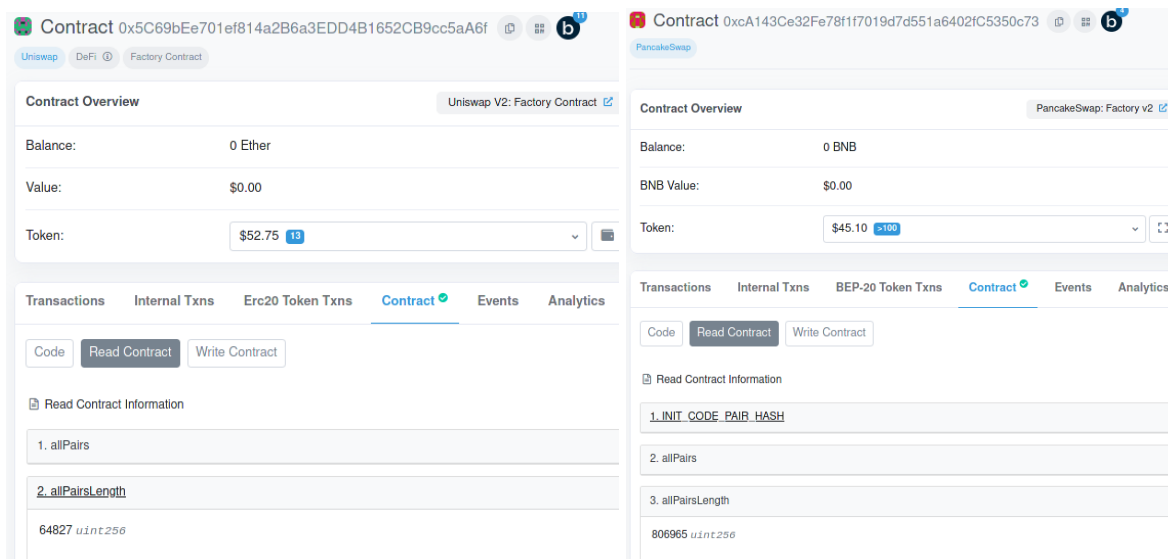


Figure 1. Left, UniSwap V2 factory¹³ indicating 64,827 token pairs have been created, and right, the PancakeSwap V2 factory¹⁴, indicating 806,945 pairs have been created.

⁷ <https://bscscan.com/tx/0x08278405d40de452ff8d2862a2937d9bc54a43c26a3dff735d56de77b0aca609>

⁸ <https://academy.binance.com/en/articles/an-introduction-to-binance-smart-chain-bsc>

⁹ <https://academy.binance.com/en/articles/what-is-bscscan-and-how-to-use-it>

¹⁰ https://ycharts.com/indicators/ethereum_average_gas_price

¹¹ <https://ethereum.github.io/yellowpaper/paper.pdf>

¹² <https://www.bsc.news/post/binance-smart-chain-experiences-congestion-from-heavy-demand>, <https://thedefiant.io/binance-smart-chain-congestion-is-slowing-down-pancakeswap/>, <https://github.com/bnb-chain/bsc/issues/189>

¹³ <https://etherscan.io/address/0x5c69bee701ef814a2b6a3edd4b1652cb9cc5aa6f#readContract>

¹⁴ <https://bscscan.com/address/0xca143ce32fe78f1f7019d7d551a6402fc5350c73#readContract>

We discuss SafeMoon specifically, due to its brief window of significant mainstream popularity (as discussed in Section 1.) providing a large amount of transactional data to analyze, as well as its novel mechanics of combining reflexive rewards (user balance passively increases over time from the transaction tax) and auto-liquidity (designed to stabilize token price). Although SafeMoon was not the first to combine token reflection and automatic liquidity, something noted by the comments in the SafeMoon contract itself¹⁵, it was the first to go ‘viral’, with search terms for SafeMoon briefly outstripping even Google searches for Ethereum, as well as its enduring market cap of roughly one billion USD¹⁶, as well as the vast number of SafeMoon ‘clones’ that were spun off from almost identical code.

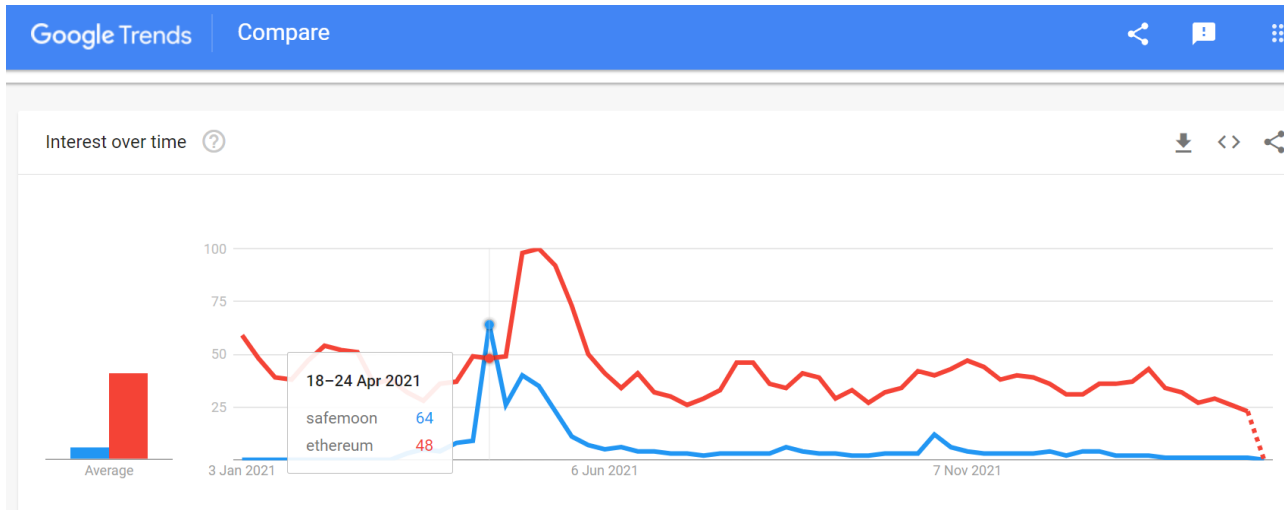


Figure 2. SafeMoon’s Google search volume score outdid that of Ethereum in April 2021.¹⁷

Likeness	Name/Symbol	Address	Network	Deployed	Diff
1.00	TEST (TEST)	0xff4f...42c3	BSC	+25d	None
0.99	EvilElonMusk (EVILELON)	0x849e...9c3d	BSC	+57d	View
0.99	MoonBonfire (MBONFIRE)	0xb9f6...2045	BSC	+55d	View
0.99	Silver (AG)	0xaf09...b106	BSC	+37d	View
0.99	t.me/SafePanantal (SAFEPAN)	0x85ff...b213	BSC	+40d	View
0.99	Safe5G (SAFE5G)	0xb53d...3f82	BSC	+48d	View
0.99	Al-Pacinu (ALPINU)	0xec70...53d7	BSC	+142d	View
0.99	SafeBullish (SBULLISH)	0x9a21...6032	BSC	+19d	View
0.99	ApolloMision (APM)	0xd128...a7eb	BSC	+57d	View
0.98	Elonzila (EZA)	0xb40f...45e9	BSC	+58d	View

WARNING: the above token(s) were flagged due to evidence of a **bug, hack, or scam**

Figure 3. Sample of contract addresses on BSC with at least 98% similarity to SafeMoon. Elon Musk, Dogs, and interplanetary spaceflight are common themes¹⁸.

¹⁵ <https://github.com/safemoonprotocol/Safemoon.sol>

¹⁶ <https://www.cryptocompare.com/coins/safemoon/overview>

¹⁷ <https://trends.google.com/trends/explore?date=2021-01-01%202022-03-15&q=safemoon,ethereum>

¹⁸ <https://tokensniffer.com/token/vlx8qkbqs33b31a0c48z1vm153mn5x0q5z06zxxdh5f405aajvbw9228v20>

2. Smart Contract Analysis of SafeMoon and its Derivatives

2.1 Gas Usage & Tax Mechanism

Despite SafeMoon’s success, the underlying SafeMoon contract is a very clear merge of the two projects it was based on: RFI (Reflect Finance)¹⁹ and, to our best knowledge, Heaven’s Gate²⁰ (although like much of the DeFI space, it’s quite probable that these token contracts are themselves iterations of previous tokens).

The result of this merge is a token that very clearly **feels** stitched together, and isn’t, for example, compliant with the ERC20 spec (e.g. missing events that are required, which we cover in more depth later). For instance, the reflective portion in RFI’s code includes essentially no comments or documentation of functionality, which make the code and what it’s doing essentially inscrutable. This frequently leads to opened issues on Github²¹ and articles like this from Medium entitled “What the hell is `_rTotal` and `_tTotal`”²².

In terms of particular functions that are problematic in SafeMoon and SafeMoon derived tokens, one of the most obvious lies in the transfer function:

```
1011     function _transfer(  
1012         address from,  
1013         address to,  
1014         uint256 amount  
1015     ) private {  
1016         require(from != address(0), "ERC20: transfer from the zero address");  
1017         require(to != address(0), "ERC20: transfer to the zero address");  
1018         require(amount > 0, "Transfer amount must be greater than zero");  
1019         if(from != owner() && to != owner())  
1020             require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");  
1021  
1022         // is the token balance of this contract address over the min number of  
1023         // tokens that we need to initiate a swap + liquidity lock?  
1024         // also, don't get caught in a circular liquidity event.  
1025         // also, don't swap & liquify if sender is uniswap pair.  
1026         uint256 contractTokenBalance = balanceOf(address(this));  
1027  
1028         if(contractTokenBalance >= _maxTxAmount)  
1029         {  
1030             contractTokenBalance = _maxTxAmount;  
1031         }  
1032  
1033         bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;  
1034         if (  
1035             overMinTokenBalance &&  
1036             !inSwapAndLiquify &&  
1037             from != uniswapV2Pair &&  
1038             swapAndLiquifyEnabled  
1039         ) {  
1040             contractTokenBalance = numTokensSellToAddToLiquidity;  
1041             //add liquidity  
1042             swapAndLiquify(contractTokenBalance);  
1043         }  
1044     }
```

Figure 4. Private `_transfer` function from SafeMoon code²³, lines 1011 to 1043.

¹⁹ <https://github.com/reflectfinance/reflect-contracts/blob/main/contracts/REFLECT.sol>

²⁰ <https://etherscan.io/address/0x054bd236b42385c938357112f419dc5943687886#code>

²¹ <https://github.com/safemoonprotocol/Safemoon.sol/issues/50>

²² <https://wd666.medium.com/what-the-hell-is-rtotal-and-ttotal-4e2ec90466f7>

²³ <https://github.com/safemoonprotocol/Safemoon.sol/blob/main/Safemoon.sol#L1011>

In this example, we can see that during the transfer function, SafeMoon checks the SafeMoon token balance held within the SafeMoon contract itself. If this balance is above the limit (set in `numTokensSellToAddToLiquidity`), the contract will attempt to sell half that number of tokens to PCS, and add the resulting BNB + SafeMoon as liquidity, for the stated reason to “decrease volatility”²⁴ by increasing the available liquidity.

While this check before adding liquidity prevents the overhead of these functions being called on every transaction, saving very approximately 250,000 gas for the adding of liquidity, and 150,000 gas for the extra swap required (*very* approx. ~\$0.80 assuming 5 Gwei and \$400 per BNB), it makes it much more difficult for crypto wallets like MetaMask or Trust Wallet to estimate the amount of gas required for the transaction to execute.

All transactions on the Ethereum (and BSC) chains require a gas ‘limit’²⁵, which is the maximum amount of gas a transaction can use before it will revert and fail. Wallets like MetaMask or Trust Wallet typically estimate the gas required for a transaction using the “`eth_estimateGas`” function of the JSON-RPC (remote procedure call) protocol²⁶. This function essentially ‘steps through’ a proposed transaction to see how much gas it will use, and provides that estimate to the user when they submit their transaction.

So, when the `swapAndLiquify` is not expected to run, the estimated gas fees will be significantly lower compared to when it *is* expected to run. In reality, however, there is a time lag between the time the user client estimates the gas used, and the time the transaction is actually ready for execution on the blockchain. During this time lag, queued transactions from other users trading SafeMoon ahead of the said user might be processed and executed. If the tax accumulated from these transactions were enough to trigger the `swapAndLiquify` function, the actual gas usage for the other user’s transaction will jump significantly. Having wrongly estimated the gas fee at the time of approval, it is likely that the said transaction will run out of gas and fail, which causes the next transaction in the queue to trigger the function, and potentially also run out of gas and fail. This difficulty in estimating gas usage is likely to be a major cause of transaction failures when trading SafeMoon or derivatives on PCS, which makes for a poor user experience, and is something we’ll be exploring shortly with an experiment.

In addition to having uncertain gas usage, SafeMoon’s attempt to use liquidity to buffer the price also has a number of issues. Primarily, the tax for SafeMoon is collected in native tokens, half of which must then be *sold* when adding liquidity. The amount that triggers `swapAndLiquify` is static, too, which meant that during May 2021 using Nomics’ token data, which put SafeMoon at an average price of \$6.57 per million tokens²⁷, a `swapAndLiquify` would sell off \$1.65 million worth USD during a `swapAndLiquify`, potentially severely impacting the price.

TipsyCoin addresses a number of these issues. First, the amount of gas used is much easier to estimate, reducing the number of failed transactions – this is because TipsyCoin *always* taxes sell transactions by taking up to 10% of the \$tipsy that would be sold, redirecting it to the TipsyCoin contract address, and then swapping it for BNB. For very small \$tipsy amounts (defined as transactions where the taxable portion to be traded for BNB would be less than 1 Gwei), the amount that would be taxed is instead burned. This means that the only transactions where gas usage cannot be easily estimated are those in the range of 15-20 Gwei, or transactions worth a fraction of a cent.

²⁴ <https://safemoon.com/whitepaper.pdf>

²⁵ <https://ethereum.org/en/developers/docs/transactions/>

²⁶ <https://eth.wiki/json-rpc/API>

²⁷ <https://nomics.com/assets/safemoon-safemoon-v1-old/history>

```

487     function transferFrom(
488         address sender,
489         address recipient,
490         uint256 amount
491     ) public noBots(recipient) returns (bool) {
492         if (!excludedFromFee[sender])
493         {
494             uint _amountBuyBack = amount * buybackFundAmount / _feeTotal/10;
495             uint _amountMarketing = amount * marketingCommunityAmount / _feeTotal/10;
496             uint _amountReflexive = amount * reflexiveAmount / _feeTotal/10;
497
498             if(_amountBuyBack + _amountMarketing > 0)
499             {
500                 uint _minToLiquify = pancakeV2Router.getAmountsOut(_amountBuyBack + _amountMarketing, _tokenWETHPath)[1];
501                 if(_minToLiquify >= 1e9) _taxTransaction(sender, _amountBuyBack + _amountMarketing, _minToLiquify);
502                 else _burn(sender, _amountBuyBack + _amountMarketing);
503             }
504
505             amount = amount - _amountBuyBack - _amountMarketing - _amountReflexive;
506             _transfer(sender, recipient, amount);
507
508             if(_amountReflexive > 0) _reflect(sender, _amountReflexive);
509
510         }
511         else
512         {
513             //Skip collecting fee if sender (person's tokens getting pulled) is excludedFromFee
514             _transfer(sender, recipient, amount);
515         }

```

Figure 5. Section of TopsyCoin's main ERC20 contract showing much more stable gas usage by taxing all sell transactions ²⁸.

While readers might assume that always taxing sells causes TopsyCoin to be more gas intensive than SafeMoon – because of a number of other gas saving techniques used in TopsyCoin, as well as the existence of gas inefficiencies in SafeMoon, the average gas required for a \$tipsy sell is still significantly lower than a SafeMoon transaction. Additionally, TopsyCoin *only* taxes when you sell, making simple transfers or buy transactions (all of which are taxed by SafeMoon and derivatives) no more expensive than standard non-transfer tax tokens, like USDC.

TopsyCoin also never tanks the price of its token when collecting tax. By only taxing sells, \$tipsy is only swapped for BNB at a time when the user was already in the process of selling their tokens. So, as an example, if a user intended to sell 100 \$tipsy for 1 BNB, the user would effectively still transfer out 100 \$tipsy, but with 10% of that output either being swapped for BNB and redirected to the tax collection addresses, like the BuyBack vault, or used for reflexive rewards. Once a significant amount of BNB has been accumulated in the BuyBack vault, the vault is then triggered by the team, and causes a net buy of tipsy. This system ensures that unlike SafeMoon and derivatives, the tax never dumps \$tipsy on the open market. Figure 6 from the tipsycoin.io website visually explains this process.

²⁸ <https://github.com/TipsyCoin/TipsyCoin/blob/main/contracts/TipsyCoin.sol#L487>

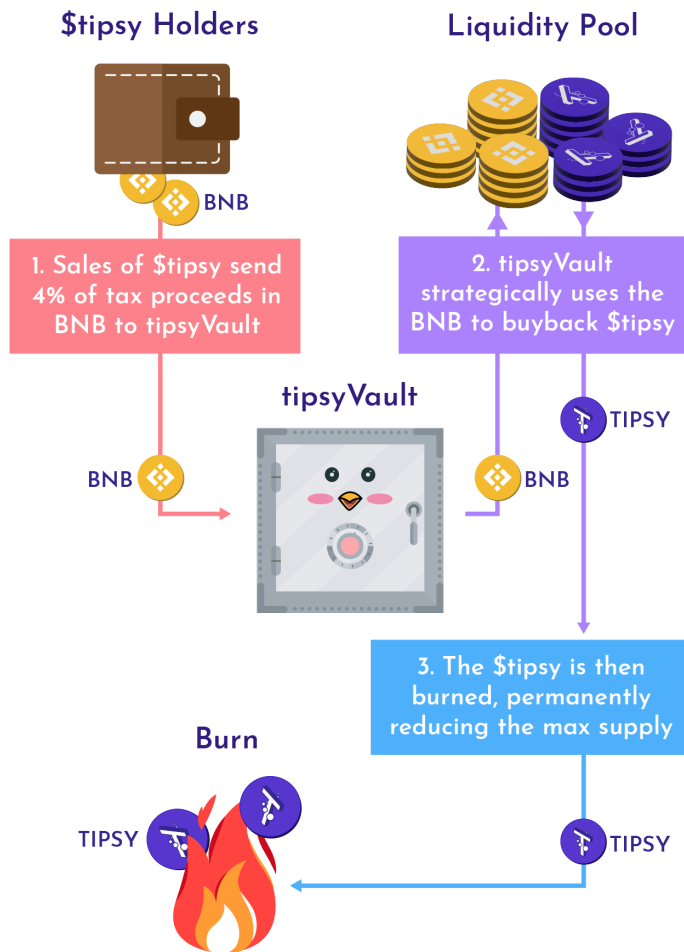


Figure 6. Infographic showing how the TopsyCoin Buyback vault (tipsyVault) works ²⁹. In addition to burning TopsyCoin, the vault is also configurable to add liquidity or distribute reflexive rewards ³⁰.

As a final comment on the add liquidity function in SafeMoon, a significant downside of adding the liquidity all at once in the manner they did, is that the potential price impact of the sell causes slippage when liquidity is added. This is generally unavoidable, and slippage is typically small, but the large size of the SafeMoon buybacks and the significant volume it attracted means that there's currently over \$2 million USD worth of BNB permanently stuck in the contract ³¹. This is another issue TopsyCoin fixes, since it allows leftover BNB to be reused during the buyback process, and other 'stuck' assets can also be salvaged from the contracts.

In summary, in this section we've hopefully explained why the unpredictable gas usage of SafeMoon and SafeMoon derivate tokens makes transactions more likely to fail. Additionally, because the tax is collected in native tokens, when the add liquidity function is eventually called, it causes a potential price drop. This price slippage combined with no salvage functions has also resulted in several million dollars being trapped in the SafeMoon contract. TopsyCoin addresses these concerns by collecting tax on every sell transaction, reducing failed transactions, as well as collecting tax proceeds in BNB. TopsyCoin also allows unspent BNB in the buyback vault to be reused, and other assets can also be salvaged if they get stuck in the TopsyCoin contracts.

²⁹ <https://tipsycoin.io/tokenomics/>

³⁰ <https://github.com/TipsyCoin/TipsyCoin/blob/main/contracts/BuyBack.sol#L868>

³¹ <https://bscscan.com/address/0x8076c74c5e3f5852037f31ff0093eeb8c8add8d3>

2.2 Reflection

Out of all the aspects of SafeMoon, its reflection code is perhaps its most novel, but also its most complex. Reflection is a way for token holders to see their balance accumulate over time purely via the reflection mechanism. Reflection is also a ‘passive’ reward system where, unlike staking or farming, no extra transactions need to be sent and instead, the user’s balance increases by holding onto the token. In addition to requiring extra transactions, traditional forms of rewards-earning, such as staking and yield farming come with several costs and risks, such as: pooling funds in potentially unverified / untrusted contracts, fees to accept rewards, and security risks with transferring funds.

While TopsyCoin implements reflection in its contract code, the concept of reflection has been used frequently in the DeFi world before as well. Projects such as EverGrow Coin have paid over \$35 million in reflection to reward its token holders, but perhaps the most notable example of reflection is the one implemented by SafeMoon—to which we will compare TopsyCoin’s implementation.

2.2.1 Reflection in TopsyCoin

Similar to SafeMoon, TopsyCoin’s reflexive rewards are funded by a tax on TopsyCoin transactions. However, as mentioned previously in the paper, unlike SafeMoon, TopsyCoin only taxes sell transactions, whereas SafeMoon taxes all coin transactions. From the 10% sell tax that is implemented in TopsyCoin, 40% of the sell tax (4% of each sale), goes towards reflection. If a token holder were to hold 0.0002% of the total supply of TopsyCoin, they would then be rewarded with 0.00002% of the 4% with reflexive awards.

Over time the burn wallet turns into the largest holder, and its earning of reflection awards as well, further decreases the total supply in circulation and thus adds to the coin’s deflation.

2.2.2 Reflection Implementation

The naive idea to implement reflection would be to simply loop through the account holders and deposit an additional number of tokens. However, this approach would be very costly in terms of gas, and instead TopsyCoin uses a deflationary ratio that increases the value of a user’s balance instead of manually adding more tokens.

The idea behind using a ratio to manage user balances, is similar to the idea of the deposit reserve ratio, where the total currency in circulation can be derived from the balance sheets of a bank. Similar to SafeMoon’s implementation, TopsyCoin uses the idea of mapping tokens in the real supply to tokens in the reflexive space, $rTotal$. The essential idea behind the mapping from real space to reflective space, is that a holder’s balance can accurately represent its share of the total supply, through the ratio of $rTotal$. Initially, when there have been no sales, $rTotal$ is mapped 1-1 to the total real supply of the coins. After each sale, the $rTotal$ multiplier is modified to represent the increase value of each coin with the following formula:

$$rTotal = \frac{totalSupply_{real} \times rTotal_0}{totalSupply_{real}} - reflectAmount$$

After calculating the $rTotal$, which represents the relationship between the real space, and the reflective space, we can effectively calculate the reflective balance a token holder will see with the following relationship:

$$userBalance = \frac{(balancereal \times r_{Total})}{e^{18}}$$

As more sales take place, the value of r_{Total} increases, which increases a holder's balance and represents their increasing share of the total supply. Let's illustrate this with an example, and for the purpose of simplicity assume the total supply of TopsyCoin was 100 tokens with all other factors kept the same. Consider the case of the first transaction from Pingu, who owns 10 coins. Another holder, Pengu has 1 coin. Pingu decides to sell all his tokens, and with 4% towards reflexive tokens that means 4 tokens must be distributed. If we use the r_{Total} formula from above, we find that the r_{Total} is now 1.04, signifying the increase in token holder's reflexive balances. If we apply this to Pengu's balance, Pengu now has 1.04 tokens, which is the same number Pengu would've had if we had simply divided 4 by 100, and distributed 0.04 to each token.

The reason this relationship works is that TopsyCoin carefully keeps track of the real supply of the token and the reflexive supply of the token. After each reflection, the total real supply is decreased by the reflected amount, however $_rTotalSupply$ always remains constant, by increasing the $_rTotal$. Therefore, through each sale of the token, the balance of a token holder is increased to represent the increase of the market share they now have.

SafeMoon implements reflection with a very similar concept of managing a relationship between real tokens and reflexive tokens, but TopsyCoin takes this idea further in several ways. Firstly, as shown in the diagram below, SafeMoon uses several variables, additional storage and convoluted functions to maintain the relationship between r_{Total} , and the total supply which they named $tTotal$. The management of so many moving parts, makes the SafeMoon code difficult to process and understand which can lead to harder code maintenance and more.

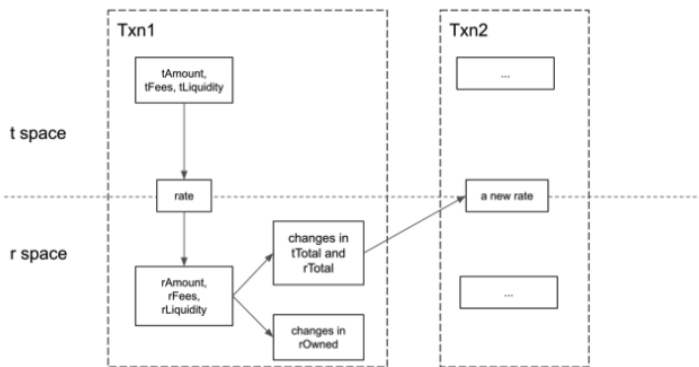


Figure 7. SafeMoon's implementation of reflection with multiple moving parts, leading to harder code maintenance ³².

TopsyCoin simply uses 2 variables: r_{Total} and $totalSupply$ to manage reflection, derived from the idea of the deposit-reserve ratio. Through reflection, we believe in rewarding our token holders for their loyalty, and we do so in clever ways.

³² <https://wd666.medium.com/what-the-hell-is-rtotal-and-ttotal-4e2ec90466f7>

2.3 Centralization Concerns

The final way we can highlight some of the technical shortcomings of SafeMoon is to discuss their audit by CertiK. The audit report indicated a number of issues discovered, many of which were not able to be addressed by the SafeMoon team, because the contract was already deployed and could no longer be altered.

Table 1. SafeMoon’s CertiK Vulnerability Summary

Vulnerability Summary

Vulnerability Level	Total	ⓘ Pending	⊗ Declined	ⓘ Acknowledged	⌚ Partially Resolved	✓ Resolved
● Critical	0	0	0	0	0	0
● Major	1	0	0	0	1	0
● Medium	1	0	0	1	0	0
● Minor	4	0	0	3	1	0
● Informational	6	0	0	6	0	0
● Discussion	1	1	0	0	0	0

SafeMoon’s summary of vulnerabilities - produced by CertiK for the SafeMoon audit³³ indicating that a number of issues remained unresolved or unmitigated.

The ‘Major’ issue at hand, is that whilst the initial liquidity of SafeMoon was locked, all of the LP tokens minted using the SafeMoon tax were sent to a single developer address. After a prolonged period, with a significant amount of tax revenue collected from the token, this provides the developers with a huge number of LP tokens (which themselves represent equal parts SafeMoon and BNB). If misused, this could result in the developers withdrawing the underlying tokens in the LP and ‘rug pulling’ them.

[SafeMoon Team]: In regards to owner control, we are a fair launch governed by a central board which is subject to governmental regulations and law. We are a legally registered entity in accordance to the law and jurisdictions in which we operate. SafeMoon is very different from other projects, and our differences provide more security for the community vs. anonymous teams and projects. Risks in regard to “rug-pulls” or anything else is mitigated due to the fact that every member of SafeMoon would be subject to litigation and likely a swift prison sentence. Additionally, outside of the law, our social lives would be in ruin, and we would not be able to show our faces in public again, let alone get another job. This should be taken into account when looking at the SafeMoon project as a whole.

Figure 8. SafeMoon’s response to SSL-04 | Centralized risk in addLiquidity in the CertiK SafeMoon audit. CertiK considered the issue ‘partially mitigated’.







³³ <https://www.certi.com/projects/safemoon>

SafeMoon claimed that they would be effectively required to carefully manage this risk, because any deviation from the path would result in swift legal action. This is rather undercut by the pending class action litigation against SafeMoon, which alleges that these assets were *not* as carefully managed as SafeMoon had assured, and that the SafeMoon team engaged in a ‘slow rug’ operation of gradually selling off their token holdings over a period of time ³⁴.

TipsyCoin fixes this major centralization problem. TipsyCoin was also audited by CertiK ³⁵. In the report, all issues classified as Minor and above were either resolved or mitigated. With security as our focus, TipsyCoin engaged in a month-long audit process with CertiK *before* launching, so that any issues identified could be discussed and addressed in plenty of time.

Table 2. TipsyCoin’s CertiK Vulnerability Summary

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Mitigated	Resolved
 Critical	0	0	0	0	0	0	0
 Major	4	0	0	0	0	2	2
 Medium	3	0	0	0	0	0	3
 Minor	1	0	0	0	0	0	1
 Informational	5	0	0	1	0	0	4
 Discussion	0	0	0	0	0	0	0

TipsyCoin’s summary of vulnerabilities, produced by CertiK for the TipsyCoin audit ³⁶ indicating that all issues Minor or above were marked as ‘Resolved’ or ‘Mitigated’.

Some readers may be concerned that the summary of the TipsyCoin CertiK audit doesn’t list all issues as being ‘Resolved’, with some instead listed as ‘Mitigated’. Rest assured that ‘Mitigated’ does not mean *unsafe*. As we have discussed in depth in our audit write-up on Medium ³⁷, the mitigated issues relate to ‘privileged’ access risk by owner or administrator accounts. Instead of running on promises that these accounts won’t be misused, the Tipsy team has taken it a step further by working with CertiK to implement both multi-sigs using Gnosis-Safe, and governance TimeLocks to manage privileged access.

This means that sensitive functions and assets are never held by a single wallet address, and are instead held in a network of decentralised smart contracts, with multi-sigs to ensure a compromised key or account doesn’t lead to the loss of user funds, with 48 hour timelocks to ensure a sufficient ‘notice period’ before any changes are made, allowing users to exit the ecosystem if they are unhappy or unsure of the impact the pending changes will bring.

Also, in our aforementioned Medium article, we have a detailed justification for why we require any level of privileged access, but the concise version is that the TipsyVerse ecosystem is rapidly expanding, and so we require some

³⁴ <https://www.classaction.org/media/merewhuader-et-al-v-safemoon-llc-et-al.pdf>

³⁵ <https://www.certik.com/projects/tipsycoin>

³⁶ <https://www.certik.com/projects/tipsycoin>

³⁷ <https://medium.com/@tipsycoin/tipsycoins-certik-audit-a5bf5afdc8a>

administrative access to the contracts to maintain compatibility with future projects like staking and the upcoming blockchain integrated game, TipsyVerse.

TipsyCoin (Post Audit) Deployment Diagram

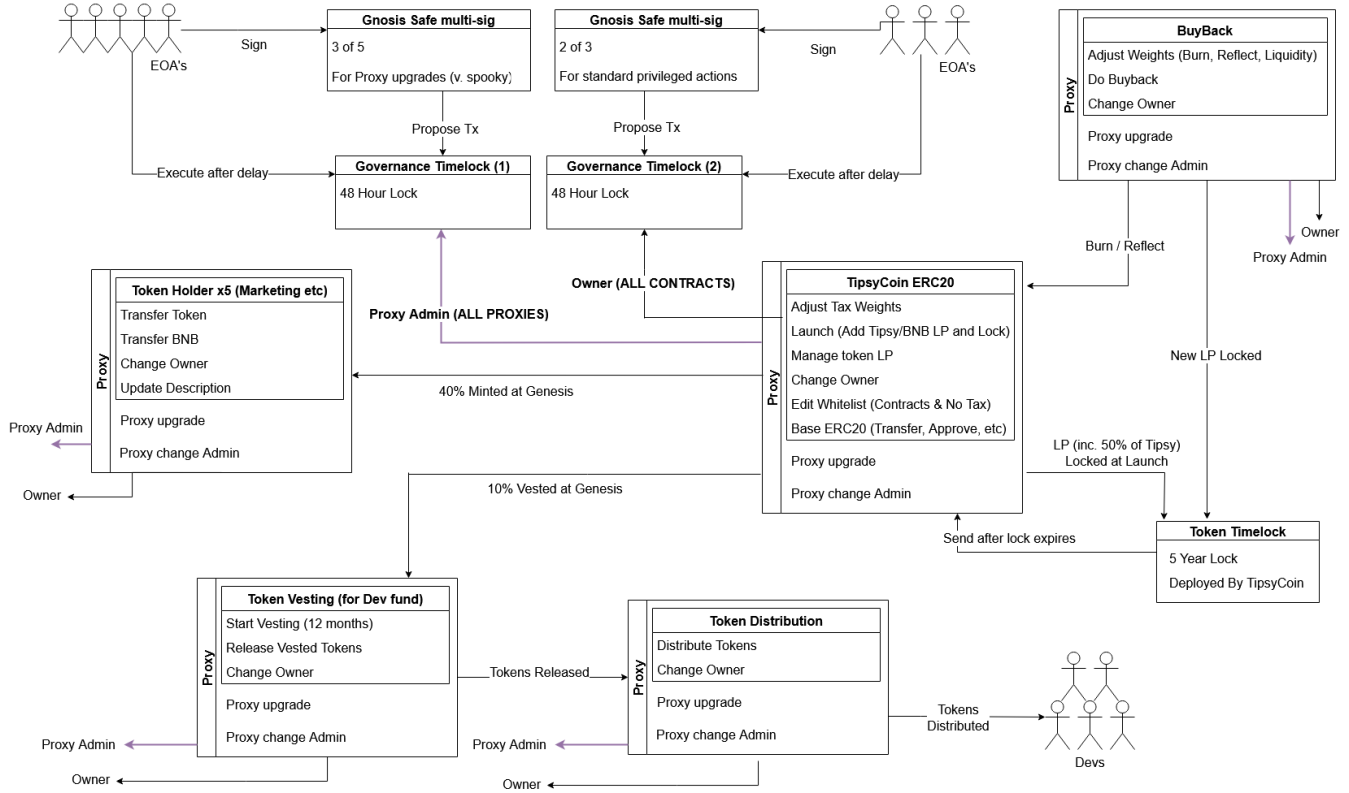


Figure 9. TipsyCoin (Post Audit) Deployment Diagram. A diagram demonstrating the network of smart contracts used to manage assets and access for TipsyCoin, ensuring a single user never has privileged access. Sourced from the TipsyCoin Github readme.md file³⁸.

In summary of the centralisation issue – SafeMoon’s audit by CertiK revealed a number of issues, many of which could not be fixed.

3. Experiments

In addition to just looking at the code and design of SafeMoon and derived tokens, no research paper would be complete without at least some experiments being undertaken. In this section, then, we collected a large number of SafeMoon transactions that occurred on the BSC, and combined our analysis from the previous sections with some more quantitative data to demonstrate how TipsyCoin takes incremental steps forward in tax on transaction tokenomics.

³⁸ <https://github.com/TipsyCoin/TipsyCoin/blob/main/README.md>

3.1 Methodology

Data was collected between blocks 5500000 and 8000000 (Between March 8th, 2021 ³⁹ and June 8th 2021 ⁴⁰) on the Binance Smart Chain. We collected this data using BSC's version of Geth ⁴¹, which is the command line interface for running a BSC node. This range includes the beginning of the SafeMoon contracts through its most active period of activity in early-mid 2021, as determined via Nomics' historical token data for SafeMoon price and volume ⁴².

The data tracked was PancakeSwap trading data between these block ranges – transfers of the token between wallets were not tracked. All transactions not 'to' the PancakeSwap router v1 or v2 (SafeMoon migrated from PancakeSwap router v1 to v2 in December 2021) were discarded, as were all transactions with function selectors ⁴³ that didn't match a PCS buy or sell operation (for example, SwapEthForTokens).

To determine whether SafeMoon was being traded on PancakeSwap, and should therefore be included in the analysis, we checked the 'path' array present in all PCS buy and sell functions that determines what assets were to be swapped. If the SafeMoon token was either at the start or end of this list, indicating that it was being bought or sold, it was added to a database of transactions to use in our analysis.

All transactions added to the database also had the following recorded: sender, blockNumber, gas used, volume of SafeMoon traded, and whether the transaction succeeded. This data formed the basis of our analysis, and included several million transactions, over a billion dollars of SafeMoon volume (at current market price) and more than 1TB worth of trading data.

3.2 Transaction Failure Rate Experiment

One of the metrics we wanted to investigate was the number of failed transactions when buying and selling SafeMoon. One of the key design decisions in TopsyCoin was the removal of a tax for buying our token, instead of only taxing sells. It is therefore important to attach some quantitative data to measure the impact of what this might have.

For this experiment, we used the dataset as described in the methodology above, and filtered it by the number of total failed transactions to get a background of what the failure rate was for trading SafeMoon. Whilst TopsyCoin has not yet launched, and so data cannot yet be directly compared, examining a small subset of USDT trading data from PancakeSwap (PCS) suggested that the UI elements in PCS, which prevent common user input errors, are able to keep failed transactions for standard coins to 1-2% on average.

3.3 Results & Discussion of Transaction Failure Rate Experiment

Table 3. SafeMoon's Block Ranges and their Failed Transaction Ratios

Block Range	Failed Transaction Ratio
5500000-6000000	3.189%
6000000-6500000	2.518%
6500000-7000000	6.924%
7000000-7500000	10.354%
7500000-8000000	4.0556%
All recorded	9.500%

³⁹ <https://bscscan.com/block/5500000>

⁴⁰ <https://bscscan.com/block/8000000>

⁴¹ <https://github.com/bnb-chain/bsc/releases>

⁴² <https://nomics.com/assets/safemoon-safemoon-v1-old/history>

⁴³ <https://docs.soliditylang.org/en/v0.8.12/abi-spec.html?highlight=function%20selector>

Our analysis of 5317904 transactions on PCS between blocks 5500000 and 8000000 (between Mar-08-2021 11:53:29 AM +UTC ⁴⁴ and Jun-04-2021 09:56:05 AM +UTC ⁴⁵) indicated that 4856750 transactions succeeded, whilst 461154 failed – this represents a failure rate of 9.50% of all transactions.

Interestingly, this failure rate was not even across the data blocks we collected. For example, between blocks 6000000 (Mar-25-2021) ⁴⁶ and 6500000 (Apr-12-2021) ⁴⁷ we observed a failure rate of 2.52%, yet between blocks 7000000 (Apr-29-2021) ⁴⁸ and 7500000 (May-17-2021) ⁴⁹, we observed a significantly increased failure rate of 10.35%.

Our results suggest an interesting trend for failing transactions, with the initial rate being low, and then quickly escalating into the double digits. We have a few possible explanations for this trend. It's perhaps likely that early adopters of SafeMoon were DeFi natives, and so were more likely to know how to set correct slippage and gas limits. Then, during the 'peak hype' surrounding SafeMoon, users were less DeFi savvy, and so did not know those options were set.

It's also possible that the way users interacted with SafeMoon changed during that time period. For example, early adopters may also have interacted more via MetaMask on a PC interface (which is quite configurable), while less experienced users may have relied on apps and internet guides, which would again, have a reduced chance of success. In addition to the actual failure rate of 9.50%, PCS's interface is also likely to have prevented a large number of 'would have failed' transactions as it performs some profiling of the transaction to determine whether it is likely to succeed.



Figure 10. PCS slippage tolerance default options. Available by clicking on the 'gear' icon on the PCS 'swap' page ⁵⁰.

Because of SafeMoon's 10% tax, PCS will prevent you from sending a transaction if this setting is less than 10%. Therefore, the actual number of failed transactions recorded on the blockchain is likely to be a vast understatement when compared to the number of transactions that were attempted but rejected by PCS's interface.

Although these 'potential transactions' are not recorded on the blockchain, and so exact numbers are likely to be impossible to determine, the large number of search results on Google, Reddit and other social media sites for the specific PCS error messages seems to suggest it was widespread. PCS also created several troubleshooting articles in an attempt to resolve 'INSUFFICIENT_OUTPUT_AMOUNT' style errors ⁵¹, with a specific guide for setting slippage to accommodate SafeMoon and similar tokens in their docs ⁵².

⁴⁴ <https://bscscan.com/block/5500000>

⁴⁵ <https://bscscan.com/block/8000000>

⁴⁶ <https://bscscan.com/block/6000000>

⁴⁷ <https://bscscan.com/block/6500000>

⁴⁸ <https://bscscan.com/block/7000000>

⁴⁹ <https://bscscan.com/block/7500000>

⁵⁰ <https://pancakeswap.finance/swap>

⁵¹ https://docs.pancakeswap.finance/help/troubleshooting#insufficient_output_amount

⁵² <https://docs.pancakeswap.finance/help/troubleshooting#issues-buying-safemoon-and-similar-tokens>

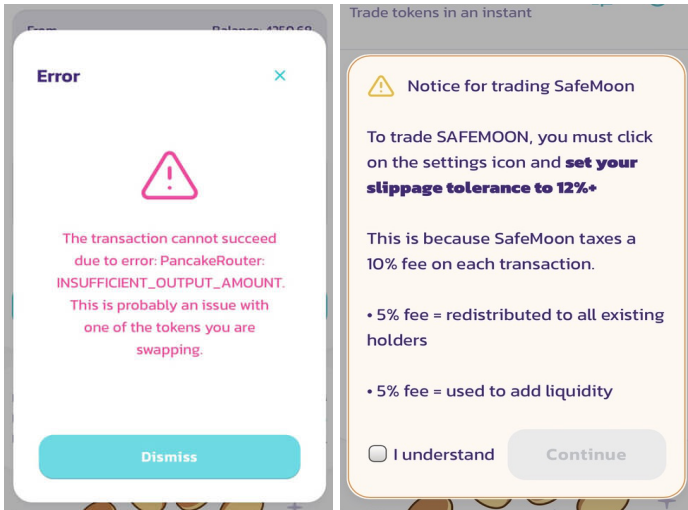


Figure 11. PCS UI demonstrating: a trade cannot occur due to an error ⁵³ (Left) , a customised error specifically when attempting to trade SafeMoon ⁵⁴ (Right).

Also, to further illustrate our previous point in section 2.1, about the gas price of SafeMoon leading to failed transactions, as well as the general gas inefficiencies of SafeMoon, we plotted the average gas used by the token, and compared it to a small sample of testnet TopsyCoin transactions.

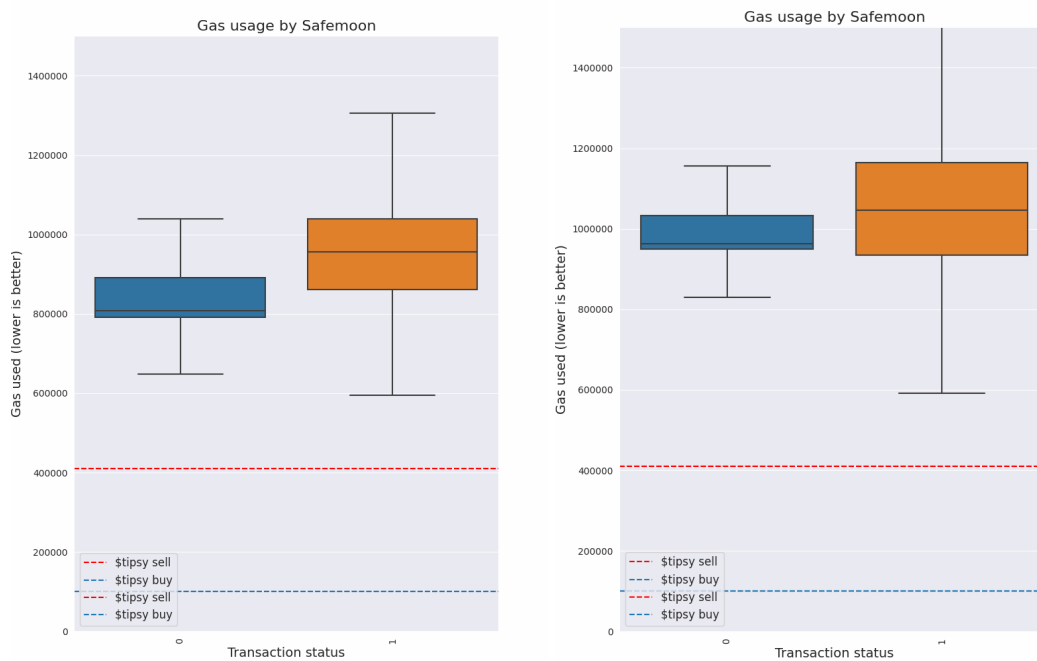


Figure 12. Average gas usage during SafeMoon transactions. Left: average gas usage from blocks 7500000 to 8000000. Right: average gas usage from blocks 8000000 to 8500000. Blue bars represent successful transactions, whilst Orange bars represent failed transactions. Dotted Red and Blue lines represent TopsyCoin transactions based on testnet data.

⁵³ <https://coinguides.org/wp-content/uploads/2021/04/pancakeswap-transaction-error.jpg>

⁵⁴ <https://community.trustwallet.com/uploads/default/original/3X/b/0/b02cc31ca10e3efaacd77c762f5c7c1e7b7a160d.jpeg>

As can be seen by Figure 14, TopsyCoin uses significantly less gas. Interestingly, the amount of gas used by SafeMoon increases over time, and the cost of failed transactions is higher as compared to the average successful transactions. This potentially indicates that the frequency of the *addLiquidity* function is increasing, and also reinforces our thoughts that failed transactions are likely because of the difficulty in estimating gas usage in SafeMoon.

3.4 Excluded Users Experiment

In this experiment, we sought to quantify how many users ‘gave up’ on purchasing SafeMoon. The data was prepared with the same methodology as described in section 3.1, except this time we filtered the dataset by unique addresses who transacted, as well as unique addresses who transacted only once, and whose sole transaction failed.

3.5 Excluded Users Results & Discussion

Table 4. Users with Failed Transactions who did not Reattempt

Block Range	Addresses with failed txs who did not reattempt (% fails)	Addresses with failed txs who did not reattempt (% all)
5500000-6000000	*	*
6000000-6500000	5.495%	0.238%
6500000-7000000	4.159%	0.460%
7000000-7500000	17.406%	2.356%
7500000-8000000	16.667%	0.904%

**Due to a data collection issue, data from the 5500000-6000000 range was not available for this experiment.*

Our results here are similar to those discovered in section 3.3, in that the number of users who had failed transactions drifted during the course of data collection. Of particular note, during the 7000000-7500000 block range, representing a date range of between Apr-29-2021 ⁵⁵ and May-17-2021 ⁵⁶, the number of users who received a failed transaction and never reattempted was over 2%.

⁵⁵ <https://bscscan.com/block/7000000>

⁵⁶ <https://bscscan.com/block/7500000>

Conclusion

In this paper we sought to explore some of the history, design features, and design issues of ‘tax on transfer’ tokens such as SafeMoon and its derivatives. To assist with our analysis, we examined the SafeMoon Github code, the CertiK audit of SafeMoon, the CertiK audit of our project, as well as collected millions of PCS transactions involving SafeMoon and conducted an experiment using this collected data, including average gas used, transaction success ratio, and whether users re-attempted their transactions on failure.

Our conclusion of SafeMoon and derivatives is that the contract code employed in such projects are unnecessarily complex and gas inefficient. Besides being inefficient, the *addLiquidity* code is only triggered on a subset of transactions, making required gas cost estimates difficult and unreliable, which leads to failed transactions and users potentially giving up on transacting the token.

We measured an average transaction failure rate of SafeMoon at 9.500%, with the gas usage increasing over time, but averaging well over 800,000 by block 7500000. We also discussed numerous CertiK audit concerns about the SafeMoon project that couldn’t be resolved because of SafeMoon’s decision to launch first and audit second. To resolve these issues, we proposed TopsyCoin, a step forward in tax on transfer tokens.

TopsyCoin is a token that is less technically complex (reduces gas fees and gas usage uncertainty); only taxes sell transactions (reduce friction in purchases, and prevent users giving up before); a BuyBack vault in BNB instead of native token (prevent price dumps when adding liquidity); improved decentralization (all LP locked for 5 years, a network of auxiliary contracts, timelocks and multi-sigs to manage privileged access); and was audited by CertiK *before* launch, allowing us to resolve or mitigate all issues tagged as ‘Minor’ or above.

TopsyCoin is the foundation that the GameFi infrastructure of TopsyVerse will be built on. The efficient reflection mechanism and key security features thus allows TopsyCoin to perform its core duties as a deflationary governance token for the game that rewards holders in the process. At this time of writing, no billion-dollar metaverse has a primary governance token that is deflationary, and so TopsyCoin resolves the issue of a crumbling economic system most play-to-earn metaverse games are faced with. In addition, TopsyCoin also has immense utility, including its users being able to pay for NFTs in \$tipsy.

Limitations

Finally, there are some limitations of this paper. Primarily, SafeMoon was migrated to a completely new ‘v2’ beginning 2021-12-31 ⁵⁷. Their new token contract was not included in our analysis and discussion, because at the time of writing, insufficient trading data was available to draw any conclusions of failure rate and gas performance. Additionally, at the time of writing, their V2 project has not been audited, making comparisons between the V1, V2, and TopsyCoin smart contracts and their audit results impossible.

⁵⁷ <https://www.bsc.news/post/safemoon-shows-how-to-migrate-tokens-from-v1-to-v2>