www.theblockchainauditor.com

SOS ERC20 Smart Contract Audit

Prepared by:
Jorge Martinez

Version: 2
Date: Sept 8, 2021

Table of Contents

1. Project Information

The SOS token is being developed in a colloborative effort between the SOS Foundation development team and the BSCPAD development team. It is an EIP20 compliant ERC20 deflationary token with community and jackpot rewards and a burn enabled by RFI reflection mechanisms. In addition, it also has Pancakeswap integration that allows for an LP fee that is used to convert sell fees in SOS into BUSD for jackpot rewards.

Lastly, this token has battle tested anti bot protection developed by the BSCPAD development team that I have audited previously.

1.1 Project Scope

We were tasked with auditing the SOS Foundation ERC20 smart contract. This audit process pertains to a github repository shared with The Blockchain Auditor on August 18th, 2021. The files within scope of this audit are:

File	MD5
./SOS.sol	81de7ffeffba4ead9f22b2dd82cecd7e

1.2 Issue Classification

Informational

This issue relates to style and security best practices but does not pose an immediate risk.

Low

An issue classified as informational does not pose an immediate threat to disruption of functionality and could not be exploited on a recurring basis, however, it should be considered for security best practices or code integrity.

Medium

An issue classified as medium has relatively small risk and isn't exploitable to circumvent desired functionality and could not have financial consequences but could put user's sensitive information at risk.

Critical

These issues in the smart contract can have catastrophic implications that could ruin your reputation, disrupt the contract's functionality, and impact the client and your user's sensitive information.

2. Findings

The objective of this audit was to identify any bugs or exploits as well as to verify the expected functionality of the SOS Foundation smart contract system. Due to the complexity of the SOS contract, a comprehensive test suite was developed to assure that the functionality was performing as expected. The smart contract architecture called for a different combination of fees for sell and buy transactions in order to reward holders and build community and jackpot funds. On buy transactions there are 2.5% community fees and 2.5% jackpot fees, and on sell transactions have a 2% jackpot fee, 2.5% jackpot fee, 1% burn, 0.5% LP fee, and a 4% holders fee that gets reflected to all holders using the RFI mechanism. To help test the fees, we added SellFee and BuyFee events.

Additionally, to help better understand and elucidiate the RFI mechanisms of the contract, we collected data from a trading simulation and generated some plots to help visualize the distribution of the fees and get a perspective on the contract state over time.

Semantic and symbolic analysis did not show any issues or warnings.

Ultimately, no issues were found in the SOS ERC20 smart contract.

0	0	0	0
Critical Issues	Medium Issues	Low Issues	Informational Issues

3. Test Results

```
  Antibot Protection
    √ owner should be the whitelister
    √ creating the LGE whitelist requires duration and amountsMax of equal length
    √ buy orders revert if not on whitelist (82ms)
    √ whitelisters cannot buy more than the specified amount max (74ms)
    √ whitelisted addresses can participate in the whitelist round (373ms)
    √ whitelist admin can add whitelist addresses using modifyLGEWhitelist (215ms)
    √ whitelist admin can modify the whitelist duration
    √ whitelist admin can modify the max tokens that can be bought during the whitelist
    √ whitelist admin can call the modifyLGEWhitelist and not change anything
    √ when the whitelist round is over, getLGEWhitelistRound returns 0
    √ whitelist admin cannot modify a whitelist that doesn't exist
    √ whitelist admin can renounce their whitelister permissions
    √ whitelist admin can tranfer their whitelisting permission to another address
    √ whitelist admin cannot transfer their whitelisting permission to the zero address

SOS Test Suite
  Deployment
    √ should be called S_
    √ should have 9 decimals
    √ should be symbolized by S
    √ should have a total supply of 1 billion
    √ should have two holders at deployment
    √ community fees should be 2.5% buy 2.5% sell with passed community address
    √ jackpot feeds should be 2.5% buy 2% sell with passed jackpot address
  Configuring tokenomics
    √ community fees should now be 3% buy 3% sell with same community address
    √ jackpot feeds should be 1% buy 1% sell with same jackpot address
    √ allows owner to set the minimum balance required to be a holder
    √ owner can set the router to a different address
    √ owner can change the lp sell percent fee
    √ owner can set the buy max amount
    √ owner can set the sell max amount
  Miscellaneous Functions
    √ included account SOS holders can use reflect() to burn a specified amount of their holdings (103ms)
    √ reflectionFromToken (44ms)
    √ can increase and decrease someone's allowance (41ms)
    √ owner can include previously excluded accounts
  Trading Fees
    √ understanding the USDT price of the first SOS token
    √ should not apply fee when token holders trade amongst themselves (132ms)
    √ should apply a 2.5% community 2.0% jackpot 0.5% lp 4% holder & 1% burn fee when selling SOS tokens (248ms)
    √ should apply a 2.5% community fee and 2.5% jackpot fee when buying SOS tokens (136ms)
  Simple Trading Simulation
    √ testing buyTokens function (143ms)
    √ testing sellTokens function (235ms)
    √ Three Buys followed by One Sell (657ms)


  39 passing (5s)

----------------------|----------|----------|----------|----------|----------------|
File                  |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------------|----------|----------|----------|----------|----------------|
 contracts\           |    96.32 |    67.86 |    96.08 |    97.35 |                |
  LGEWhitelisted.sol  |      100 |    82.35 |      100 |      100 |                |
  SOS5.sol            |    95.63 |    61.54 |    95.24 |     96.8 |... 470,554,558 |
 contracts\interfaces\|      100 |      100 |      100 |      100 |                |
  IERC20.sol          |      100 |      100 |      100 |      100 |                |
  interfaces.sol      |      100 |      100 |      100 |      100 |                |
----------------------|----------|----------|----------|----------|----------------|
All files             |    96.32 |    67.86 |    96.08 |    97.35 |                |
----------------------|----------|----------|----------|----------|----------------|
```

4. Contract State Data Analysis

4.1 Token Price vs. txCount

Figure 4.1 Token Price vs txCount

In figure 4.1 we plotted out the token price over time as groups traders were steadily added over time and previous sellers were selling on top of their buys.

4.2 jackpotBalanceSOS vs. txCount

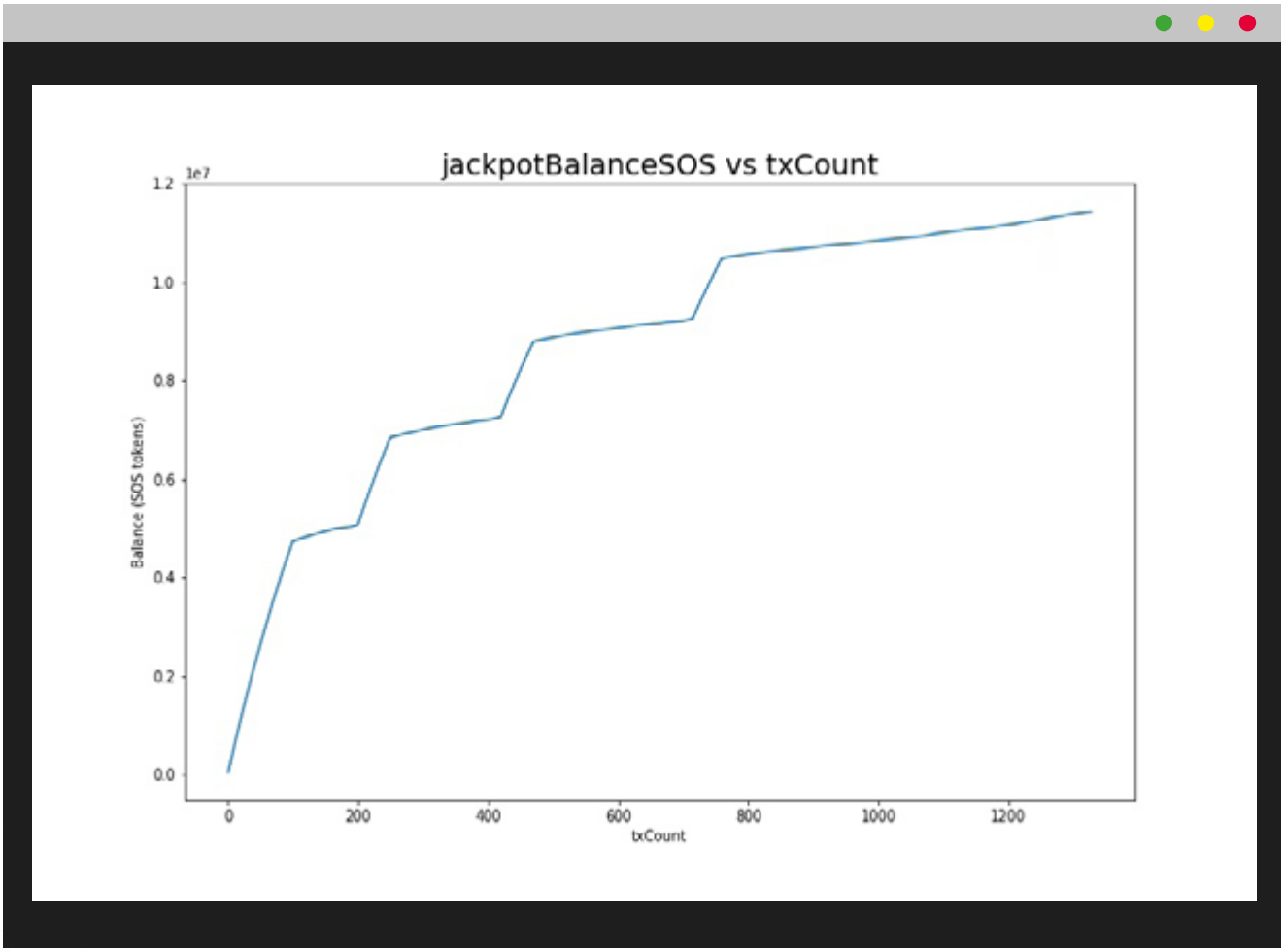

Figure 4.2 jackpotBalanceSOS vs txCount

Figure 4.2 shows us that the jackpot balance is accumulating SOS over time. In a little less than two thousand transactions, over 12 million SOS were sent to the jackpot balance. The amount that is accumulated over time is largely dependent on the amount of liquidity added and ratio of the tokens.

4.3 deadAddressBalanceSOS vs. txCount

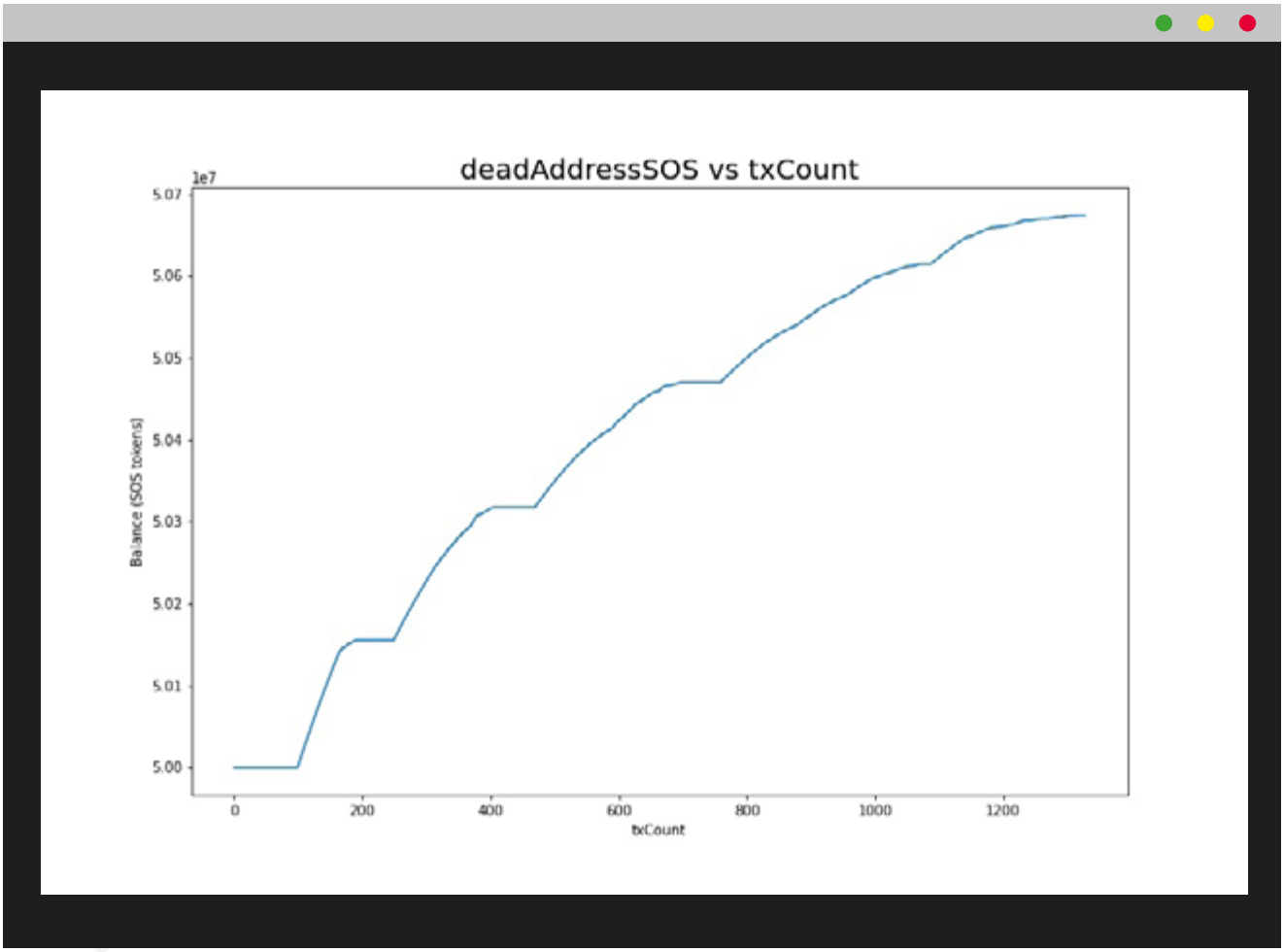

Figure 4.3 deadAddressBalanceSOS vs. txCount

Figure 4.3 show us to how the deadAddress is burning the reflect fee it gets for owning 5% of the supply. In less than one thousand five hundred transactions it was able to accumulate close to 1 million tokens in addition to the initial 50 million SOS tokens that were added for liquidity in this

4.4 communityBalanceSOS vs. txCount

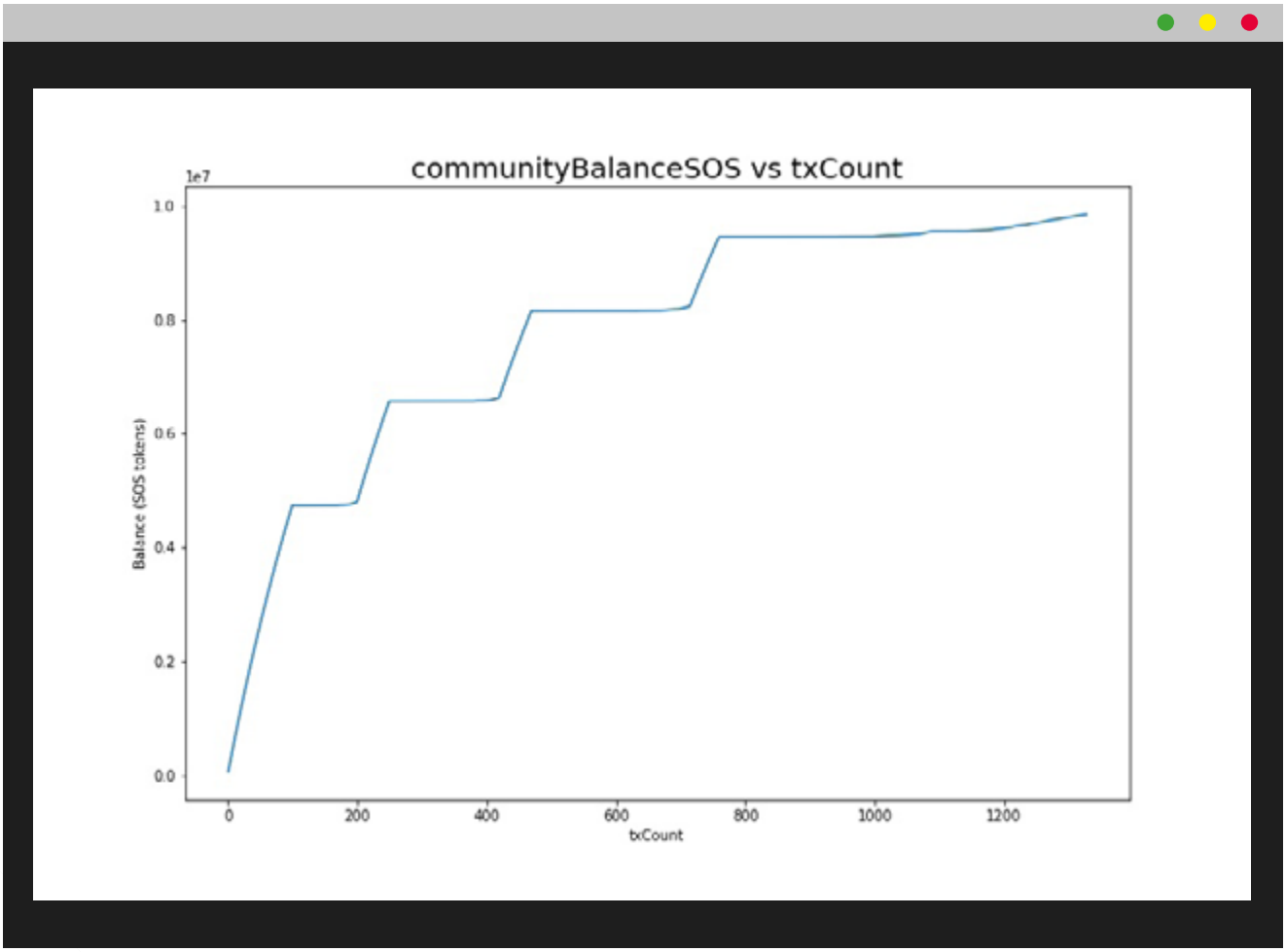

Figure 4.4 communityBalanceSOS vs. txCount

In Figure 4.4, we can observe the community address SOS balance accumulating almost 10 million SOS tokens in less than one thousand five hundred transactions. In order to give a reference based on the liquidity that was added, in the next page you can see the total USDT amount accrued at the community address over the same timespan.

4.5 communityBalanceUSDT vs. txCount

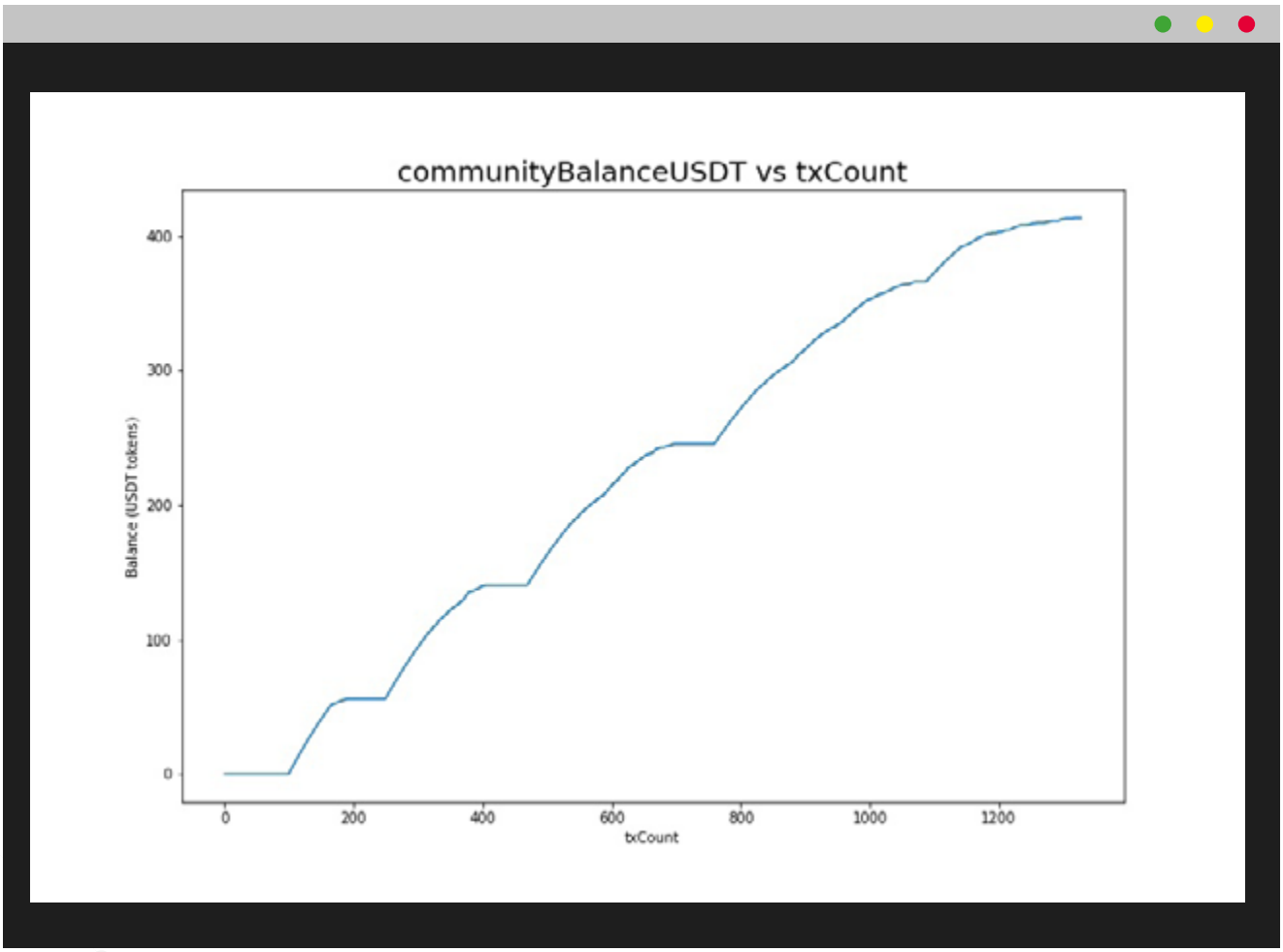

Figure 4.5 communityBalanceUSDT vs. txCount

As opposed to Figure 4.4, Figure 4.5 shows the community USDT balance after almost one thousand five hundred transactions. We can see here that it accumulated about 400 USDT over two thousand transactions. At this rate, it wouldn't take long for the community and jackpot addresses to acquire a sizeable amount of rewards.

4.6 contractBalanceSOS vs. txCount

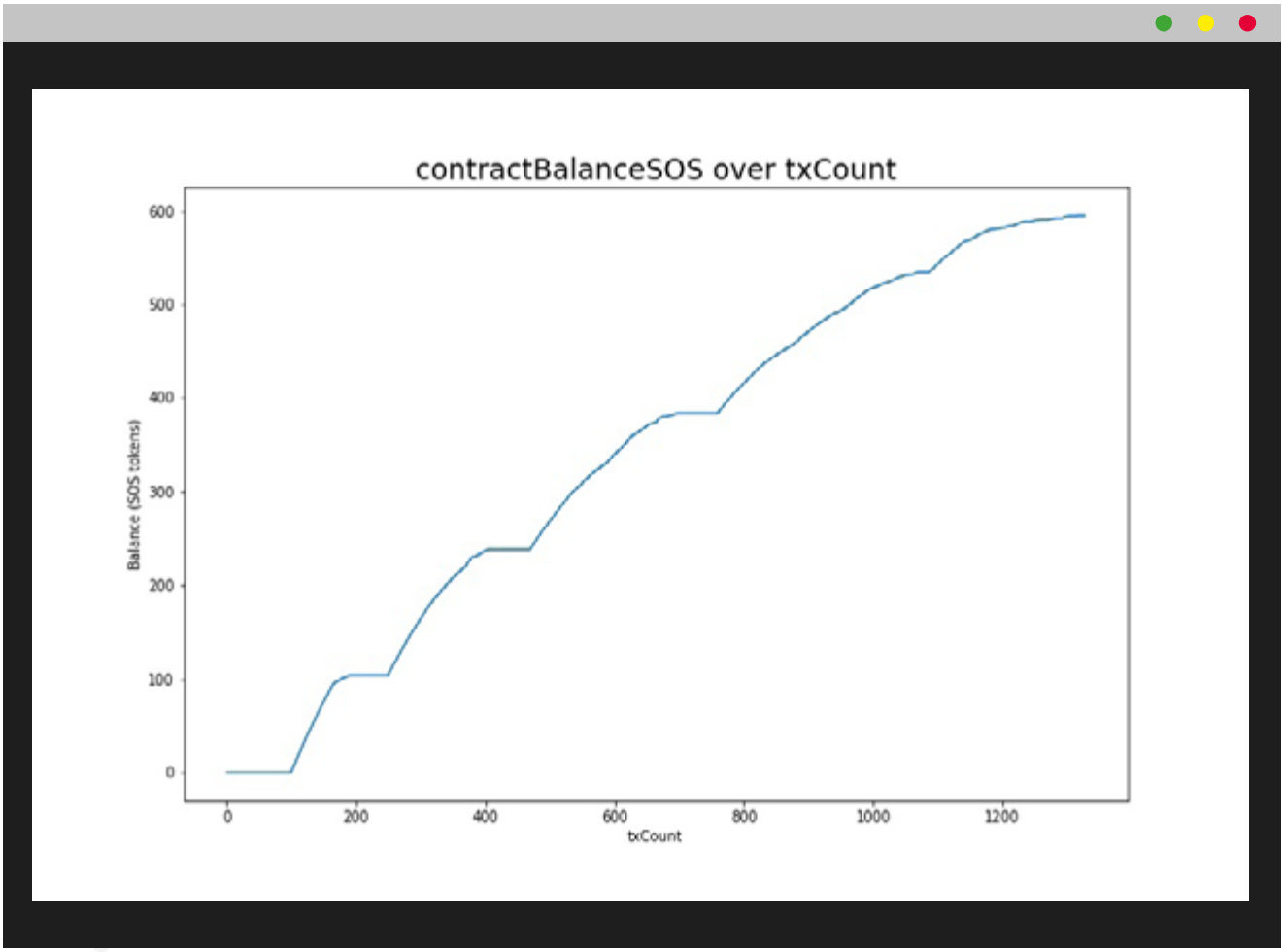

Figure 4.6 contractBalanceSOS vs. txCount

Lastly, Figure 4.6 is an important plot because it shows how a minimal amount of SOS is accumulated at the contract address . This is caused by a residual leftover from adding liquidity in every sell transaction. Although this may seem alarming, it is not a concern as the amount of SOS accumulating is insignificant. As seen in Figure 4.4, over the same timespan over 9 Million SOS tokens were accummulated in the community address; as such, the SOS amount accruing in the contract address is almost atomical. Regardless, the SOS team added a rescueToken() function in order to remove the tokens from the contract address and send them either to jackpot or community addresses at a later time.

The SOS and BSCPAD teams took every precaution within reason to secure their smart contract from bugs and exploits.
In addition, anti-bot measurements were deployed preventing bad actors from preying on investors.The Blockchain Auditor commends the SOS Foundation and the BSCPAD team for their high code standards and looks forward to the launch of the SOS Foundation and what humanitarian efforts it can facilitate in the world.

The Blockchain Auditor