Super Sushi Samurai Token

Security Review

Cantina Managed review by:

0xRajeev, Lead Security Researcher

r0bert, Security Researcher

Sujith Somraaj, Associate Security Researcher

April 20, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Super Sushi Samurai is a social strategy focused idle fully on-chain game, played on the telegram app and powered by the Blast network

From Apr 1st to Apr 3rd the Cantina team conducted a review of sss-token-contracts on commit hash f5fa785b. The team identified a total of **18** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 2
- Low Risk: 7
- Gas Optimizations: 0
- Informational: 9

3 Findings

3.1 Medium Risk

3.1.1 Vesting of developer tokens is excessively slow

Severity: Medium Risk

Context: SSS.sol#L232-234

Description: The full vesting of developer SSS tokens is contingent upon reaching a trading volume that is 160 times the `TOTAL_SUPPLY` of SSS tokens. To evaluate the feasibility of this condition, a test was designed and conducted as follows:

- **Liquidity Pool Reserves:** 10,000 WETH and 80% of SSS `TOTAL_SUPPLY`.
- **Test Mechanism:** Conduct a swap of 500 WETH for SSS, followed by the reverse swap (SSS for WETH), repeatedly. Each swap operation moves 5% of the total WETH reserves in the liquidity pool.
- **Evaluation Criterion:** Count each round of the swap pair (WETH $\rightarrow$ SSS and SSS $\rightarrow$ WETH) as a single loop execution.
- **Outcome:** After 25,915 such loop executions (equivalent to 51,830 swap operations), the `devTokenAmountRemain` state variable showed a decrease from 20,681,473,014,663.449761394484926418 to 18,803,903,272,867.618094644255968012, indicating approximately a 10% reduction in the vested tokens due to over 50,000 high-volume transactions.

Impact: The dev SSS tokens will never be fully vested. This test demonstrates that even with significant and high-volume trading activity, the developer SSS tokens do not fully vest, falling short by a considerable margin.

Likelihood: High + Impact: Low (as the tokens can be rescued after 1 year) = Severity: Medium.

Recommendation: Consider decreasing the `targetVolume` in order to allow the dev SSS tokens to fully vest.

SSS: Acknowledged.

Cantina: Acknowledged.

3.1.2 Privileged role and actions across SSS token logic lead to centralization risks for users

Severity: Medium Risk

Context: SSS.sol

Description: Several `onlyOwner` functions across SSS token logic affect critical protocol state and semantics, and therefore, lead to centralization risk for users. Some examples are highlighted below:

1. `setRestrictedTradingPool()` can arbitrarily include addresses as `restrictedTradingPools` to censor them from token transfers as enforced in `_preCheck()` of the overridden `_update()` function.
2. `rescueToken()`, which is meant to:
    1. Rescue any accidentally sent non-SSS tokens at any time or
    2. Rescue accidentally sent SSS tokens after 1 year of deployment, allows the owner to potentially access unclaimed community tax tokens and dev tax+unlocked tokens which continue to accumulate in the SSS token contract after 1 year.
3. `setCommunityAddress()` can arbitrarily change the `communityAddress`, which can claim community tax tokens via `claimCommunityTax()`.
4. `setDevAddress()` can arbitrarily change the `devTaxReceiverAddress` and `devTokenReceiverAddress`, which receive dev tax and dev unlocked tokens via `claimDevTax()` and `claimDevToken()`.
5. `setLiquidityPool()` can arbitrarily change an address's protocol-recognized `liquidityPools` status, which impacts its consideration in tax calculation and unlocking dev tokens.

6. `changeTaxPercent()` can change the default values of `buyTaxPercent` (2%) and `sellTaxPercent` (2%) to a maximum of 5% and the default split-up of `devPercent` and `communityPercent` from 20-80 to any percentage of choice.
7. `setPreMigrationOperator()` can arbitrarily include addresses that may trade in SSS tokens before migration is complete.
8. `setExcludeFromTax()` can include the liquidity pool address and prevent the developer's and community taxes from being charged.
9. `addLiquidity()` can be used to add dev tax tokens and community tax tokens held in the token contract to be added to liquidity pool.
10. `setExcludeFromTax()` can arbitrarily include/exclude any address for tax payments during transfers.

Impact: If the privileged role is compromised, it can arbitrarily affect critical protocol-wide state and semantics. Likelihood: Low + Impact: High (Loss/lock of tokens in the worst-case scenario) = Severity: Medium.

Recommendation: Consider:

1. Documenting the privileged role and actions for protocol user awareness.
2. Enforcing role-based access control, where different privileged roles control different protocol aspects and are backed by different keys, to follow the separation-of-privileges security design principle.
3. Enforce reasonable thresholds (e.g., with a split between `devPercent` and `communityPercent`) and checks wherever possible.
4. Emitting events for all privileged actions, e.g., `configBlastPointsOperator()`.
5. Privilege actions affecting critical protocol semantics should be locked behind timelocks so that users can decide to exit or engage.
6. Following the strictest opsec guidelines for privileged keys, e.g., use of reasonable multi sig and hardware wallets.

SSS: Acknowledged.

Cantina: Acknowledged.

3.2 Low Risk

3.2.1 Missing `address(this)` check in `setExcludeFromTax()` may lead to unexpected taxation

Severity: Low Risk

Context: SSS.sol#L288-L295

Description: The function `setExcludeFromTax()` excludes an address from being taxed while transferring. In the constructor, the token contract itself is excluded from being taxed. Hence, any transfers from/to the token contract are tax-free. However, the owner can accidentally reverse this by including `address(this)` for taxes.

Impact: This leads to double the taxes being charged for `dev tax` and `community tax` because of the additional tax when they are claimed. This also leads to taxes being charged when contract owner/operator transfers the initial DEX+DEV supply to the token contract during deployment.

Likelihood: Very Low + Impact: Medium = Severity: Low.

Recommendation: Consider adding a check to ensure that `address(this)` is always excluded from taxes.

```
  function _setExcludeFromTax(address account, bool exclude) internal {
+   if (account == address(this) && !exclude) revert();
    excludeFromTaxes[account] = exclude;
    emit SetExcludeFromTax(account, exclude);
  }
```

SSS: Addressed in d41e6166.

Cantina: Reviewed that this is fixed by adding `require(account != address(this), "Cannot exclude contract address");` to function `setExcludeFromTax`.

3.2.2 Reconfiguring Blast points operator via `configBlastPointsOperator()` will lead to unexpected behavior

Severity: Low Risk

Context: SSS.sol#L113, SSS.sol#L344-L346, Blast Documentation

Description: Given that `IBlastPoints(blastPointAddress).configurePointsOperator(blastPointOperator);` is already called in the constructor (as recommended in the Blast documentation), the `configBlastPointsOperator()` setter is presumably to change the existing operator. However, according to Blast docs:

> Once the points operator has been set once, only the existing points operator can update it by calling `configurePointsOperatorOnBehalf`. function configurePointsOperatorOnBehalf(address contractAddress, address operator) external; The first parameter of configurePointsOperatorOnBehalf is the contract you'd like to change the points operator for. The second parameter is the new points operator. The msg.sender must be the current points operator. This mechanism allows you to change your points operator at any point after your contract has been created.

Therefore, attempting to reconfigure blast points operator via `configBlastPointsOperator()` will lead to unexpected behavior, i.e. will likely revert (Blast docs do not cover this scenario) or not achieve operator reconfiguration as expected because `msg.sender` is the `sss` token contract and not the previously set `blastPointOperator`.

Impact: Blast points will not be distributed to the new operator and may be lost.

Likelihood: Low (Reconfiguration may not be required) + Impact: Low (Previously set `blastPointOperator` will have to reconfigure the new operator by calling `configurePointsOperatorOnBehalf()`) = Severity: Low.

Recommendation: Consider replacing `configBlastPointsOperator()` with an alternative mechanism.

SSS: Acknowledged.

Cantina: Acknowledged.

3.2.3 Lack of two-step ownership transfer is risky

Severity: Low Risk

Context: SSS.sol#L16

Description: The ownership transfer process for contracts inheriting from `Ownable` involves the current owner calling `transferOwnership()` function. If the nominated EOA account is not a valid account, it is possible that the owner may accidentally transfer ownership to an uncontrolled account thereby losing access to all functions with the `onlyOwner` modifier.

`SSS.sol` has several `onlyOwner` functions and uses `Ownable`.

Impact: All `onlyOwner` functions may become unusable.

Likelihood: Very Low + Impact: High = Severity: Low

Recommendation: It is recommended to implement a two-step ownership transfer where the owner nominates a new owner and the nominated account explicitly accepts ownership. This ensures the nominated EOA account is a valid and active account. This can be achieved by using OpenZeppelin's Ownable2Step instead of `Ownable`.

SSS: Addressed in d41e6166.

Cantina: Reviewed that this is fixed as recommended by using Ownable2Step.

3.2.4 `_update()` considering `0xdead` as burn address will lead to missed taxes

Severity: Low Risk

Context: SSS.sol#L130-L135

Description: The `_update()` internal function in `SSS.sol` contract overrides the inherited `_update()` function from the base `ERC20.sol` contract of the Openzeppelin library.

While the overridden function in `SSS.sol` contract considers sending tokens to either `address(0)` or `0xdead` as valid burn operations, the function in ERC20 only treats `address(0)` as a valid burn address.

As a result, if tokens are transferred to the `0xdead` address, it is considered a burn operation by SSS, but the total supply is not reduced, leading to a mismatched state between SSS and ERC20.

```
function _update(address from, address to, uint256 amount) internal override virtual {
   // ...
   if (from == address(0) || to == address(0) || to == address(0xdead)) {
       super._update(from, to, amount);
       return;
   }
   // ...
}
```

Impact: If users transfer to the `0xdead` address, it is considered a burn by SSS, but the total supply is not reduced thereby affecting the overall accounting. On the other hand, if supply is not reduced then such transactions should be taxed as developers and the community will miss taxes.

Likelihood: Low (Token burns to `0xdead` should be rare) + Impact: Low (Missed taxes on token transfers to `0xdead`) = Severity: Low.

Recommendation: Consider sending tokens only to `address(0)` as a valid burn operation by removing `address(0xdead)` from consideration in `_update()`.

SSS: Addressed in d41e6166.

Cantina: Reviewed that this is fixed as recommended.

3.2.5 `rescueETH()` may revert silently

Severity: Low Risk

Context: SSS.sol#L334

Description: The `rescueETH` function is a privileged function that allows the owner to move native tokens from the `SSS token` contract to itself. However, this function could be optimized, as `send` will forward ETH with 2300 GAS, which might fail if the owner is a multi-sig (or) has any other privileged actions in the fallback receive function.

```
function rescueETH(uint256 amount) external onlyOwner returns (bool success) {
   success = payable(msg.sender).send(amount);
    emit RescueETH(amount, success);
}
```

Impact: Owner address cannot be a Gnosis multi-sig (or) a contract that consumes more than 2300 GAS while receiving native tokens.

Likelihood: Low + Impact: Low = Severity: Low

Recommendation: Use `call` to transfer ETH to the owner instead of `send`. Additionally, we could also consider adding a recipient parameter for more flexibility.

```
function rescueETH(uint256 amount) external onlyOwner returns (bool success) {
   (success, ) = payable(msg.sender).call{value: amount}("");

   if(!success) revert();
   emit RescueETH(amount, success);
}
```

SSS: Addressed in d41e6166.

Cantina: Reviewed that this is fixed as recommended.

3.2.6 First SSS buyer from the WETH/SSS liquidity pool can always sell in the future for a profit

Severity: Low Risk

Context: SSS.sol#L108

Description: As mentioned in the documentation the project's revenue model relies exclusively on a trading tax applied within a single DEX. The initial liquidity pool for this DEX is established by depositing 80% of the SSS `TOTAL_SUPPLY` and an Ether amount setting the initial SSS token price at $0.000000001125, as detailed in the tokenomics section of the documentation.

However, this implementation introduces the following issue:

- Initial Purchase Impact: The first transaction to buy SSS tokens from the DEX increases the SSS price by contributing WETH and withdrawing SSS, thus depleting the SSS reserves and inflating the price.
- Subsequent Transactions: As more users buy SSS, the imbalance between WETH and SSS reserves grows, driving the price of SSS even higher.
- Exploitation Strategy: The initial buyer can exploit this by conducting what could be called a multi-block sandwich attack: purchasing a significant portion of the SSS reserve at the initial lower price, waiting for others to further drive up the price, and then selling the initially acquired SSS at a substantial profit.

Impact: First buyer can secure a disproportionate share of SSS reserves at a low cost to thereby inflate the entry price for subsequent buyers and later sell for significant profit. The planned removal of per user/transaction trade limit and bot detection logic may increase the likelihood of this scenario.

Likelihood: Low + Impact: Med = Severity: Low.

Recommendation: Consider a revision of the token launch strategy with one potential option being to conduct a presale of SSS tokens at a fixed price before establishing the liquidity pool. Consider retaining the per user/transaction trade limit and bot detection logic.

SSS: Acknowledged.

Cantina: Acknowledged.

3.2.7 SSS token contract may become insolvent to cover `devTokenAmountClaimable` claims

Severity: Low Risk

Context: SSS.sol#L208-210

Description: Upon deployment, the `SSS` contract calculates initial tax allocations as follows:

- Developer Tax Allocation: `devTaxTokenAmountAvailable` is set to 0.4% of the total trading volume (`tradingVol`).
- Community Tax Allocation: `communityTaxTokenAmountAvailable` is set to 1.6% of the total trading volume.

These allocations are intended to increase with each taxable transaction, as delineated in:

- For general tax application: SSS.sol#L130-150.
- For specific increase logic: SSS.sol#L207-L224.

However, the 5% of total SSS token `DEV_SUPPLY` is not enough to cover all the `devTokenAmountClaimable` claims because it does not account for the initial `devTaxTokenAmountAvailable` and `communityTaxTokenAmountAvailable` allocations for 2% tax over the carried over trading volume (in the constructor), which is performed as part of the v1 to v2 migration.

Impact: A shortfall in the contract's ability to dispense owed SSS tokens to developers or the community, due to over-allocation without corresponding tax collection.

Likelihood: Very low + Impact: Medium = Severity: Low.

Recommendation: Consider sending the `SSS` contract more than 5% of the total SSS token supply to cover tax claims for the trading volume carried over from v1 as part of migration.

SSS: Acknowledged.

Cantina: Acknowledged.

3.3 Informational

3.3.1 Front-running risks are present if contract is deployed on other blockchains

Severity: Informational

Context: SSS.sol#L174-L175, SSS.sol#L177

Description: The `SSS` contract presents multiple front-running risks that can not be exploited in the Blast blockchain due to the centralization layer added by the sequencer. However, if the `SSS` contract is deployed on another chain where front-running is possible, then these risks should be taken into consideration:

1. Approval race condition present in ERC20 standard implementations (i.e. no increase/decrease allowance methods).
2. Use of `block.timestamp` as deadline in `_addETHLiquidity()` function: Most AMMs allow passing a deadline parameter. Protocols typically set it to `block.timestamp`, which does not protect the swap in any way because the validator can hold the transaction as the `block.timestamp` recorded in the transaction will reflect the block's time at insertion.
3. No slippage prevention in `_addETHLiquidity()` function which sets `amountAMin` and `amountBMin` to 0.

Impact: Loss of funds when adding liquidity to the liquidity pool + possible users approving amounts higher than intended.

Likelihood: Very unlikely + Impact: Medium = Severity: Informational.

Recommendation: Consider the risks mentioned in case `SSS` token contract is deployed on a different blockchain.

SSS: Acknowledged.

Cantina: Acknowledged.

3.3.2 Missing revocation can lead to residual approval for Uniswap router while adding liquidity

Severity: Informational

Context: SSS.sol#L171

Description: `addLiquidity()` adds liquidity to the Uniswap pool for SSS-WETH pair by utilizing the SSS Token and WETH present in the token contract. In this process, the router contract is given the necessary token approval. However, there is no guarantee that the Uniswap router will utilize the entire token amount.

Impact: There may be be some residual approval for the Uniswap router from the token contract.

Recommendation: Consider revoking token approvals after adding liquidity because SSS tokens used for multiple reasons are pooled in the token contract.

```
  function _addETHLiquidity(uint256 ethAmount, uint256 tokenAmount) internal {
     _approve(address(this), address(uniswapV2Router), tokenAmount);
     // ...
+    _approve(address(this), address(uniswapV2Router), 0);
  }
```

SSS: Addressed in d41e6166.

Cantina: Reviewed that this is fixed as recommended.

3.3.3 Miscalculations in migration from v1 to v2 may lead to unexpected behavior

Severity: Informational

Context: Global scope

Description: The protocol is expected to perform a token migration from v1 to v2 as part of their relaunch given the prior incident with their v1. This involves minting the entire token supply to the token contract deployer i.e. owner/operator. The deployer/owner/operator is then expected to transfer v2 tokens to v1 holders as determined by their snapshot at the time of incident. They are also expected to migrate other data/tokens (e.g. trading volume) from v1 and add appropriate liquidity to a UniswapV2 pool.

While some of the above migration steps have been summarized in their documentation/communication, the migration script itself has not been made available and is therefore out-of-scope for this review.

Impact: Any miscalculation in token holdings/transfers or misconfiguration of the relaunch setup may lead to unexpected behavior.

Recommendation: Token/data migration from v1 to v2 should be performed with caution considering the different token holders, supplies and the required setup for relaunch.

SSS: Acknowledged.

Cantina: Acknowledged.

3.3.4 Missing checks on `initPool` parameters allow incorrect pool initialization

Severity: Informational

Context: SSS.sol#L242-L247

Description: According to the contract docs, 80% of the total supply is added to a Uniswap liquidity pool alongside $500k worth of ETH during the contract's initialization. But there are no bounds/checks to ensure that `initPool()` is called with such values.

Impact: Owner can accidentally initialize the Uniswap pool with different values.

Recommendation: Consider making the `tokenAmount` parameter in `initPool()` equal to the `DEX_SUPPLY` and the `ethAmount` is equivalent to $500K or more using an oracle. Or if there is any other business logic to add the 80% supply to liquidity pool, consider documenting it accordingly.

SSS: Acknowledged.

Cantina: Acknowledged.

3.3.5 `addLiquidity()` will revert if contract's balance is less than `ethAmount`

Severity: Informational

Context: SSS.sol#L169

Description: The owner uses `addLiquidity()` to add liquidity for the SSS-WETH Uniswap pair. This function is non-payable and so the owner is expected to have sent the necessary `ethAmount` to the contract before executing this call.

However, there is no validation to ensure that contract's balance is greater than `ethAmount` to process the transaction.

Impact: Function will revert if contract's balance is less than `ethAmount`.

Recommendation: Consider adding a balance check before calling `addLiquidityETH` on Uniswap.

```
  function _addETHLiquidity(uint256 ethAmount, uint256 tokenAmount) internal {
+   if (address(this).balance < ethAmount) revert();
    _approve(address(this), address(uniswapV2Router), tokenAmount);
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
      address(this),
      tokenAmount,
      0, // accept any amount of ETH
      0, // accept any amount of token
       address(this),
       block.timestamp
   );
  }
```

SSS: Addressed in d41e6166.

Cantina: Reviewed that this is fixed as recommended.

3.3.6 Code duplication in `_unlockTokenForDev` and `_calculateUnlockTokenForDev` is unnecessary

Severity: Informational

Context: SSS.sol#L227

Description: The function `_calculateUnlockTokenForDev` in `SSS.sol` calculates the unlocked amount based on the trade volume during a transfer action (based on specific criteria, including whether the sender/receiver is a trading pool). This function is invoked in two places: the `constructor` and `_unlockTokenForDev`. It has an early return optimization when `devRemainToken == 0`.

However, when called from the constructor, it doesn't need this check because the value is initialized with 5% of the total supply. Similarly, in `_unlockTokenForDev`, this check is already performed before calling '_calculateUnlockTokenForDev' which makes this check redundant.

```
function _unlockTokenForDev(address from, address to, uint256 amount) internal {
  // ...
  uint256 devRemainToken = devTokenAmountRemain;
  if(devRemainToken == 0) {
     return;
   }

  uint256 unlockAmount = _calculateUnlockTokenForDev(amount);
  // ...
}

function _calculateUnlockTokenForDev(uint256 amount) internal view returns (uint256) {
  uint256 devRemainToken = devTokenAmountRemain;
   if(devRemainToken == 0) {
      return 0;
    }

   // ...
}
```

Recommendation: Because the `devRemainToken == 0` check in `_calculateUnlockTokenForDev` is not reachable in any condition, consider removing it.

SSS: Addressed in d41e6166.

Cantina: Reviewed that this is fixed by refactoring `_unlockTokenForDev` to remove the check there.

3.3.7 `_preCheck` function has unused `amount` parameter

Severity: Informational

Context: SSS.sol#L152-L162

Description: The internal function `_preCheck` in `SSS.sol` validates whether migration is completed. If not, it prevents user transfers during the migration phase and the trading of tokens on restricted trading pools.

However, this function accepts an additional `amount` parameter, which is unused.

Recommendation: Consider removing the unused `amount` parameter from `_preCheck` function.

SSS: Addressed in d41e6166.

Cantina: Reviewed that this is fixed as recommended.

3.3.8 Redundant `devTokenAmountRemain` initialization in constructor is unnecessary

Severity: Informational

Context: SSS.sol#L52, SSS.sol#L118

Description: The variable `devTokenAmountRemain` tracks the total supply of dev tokens that have yet to be claimed through dev tax. This variable is set to `DEV_SUPPLY` during its declaration; however, it is again initialized to the same value inside the constructor, which is redundant.

Recommendation: Consider removing the redundant initialization of `devTokenAmountRemain` in the constructor.

SSS: Addressed in d41e6166.

Cantina: Reviewed that declaration removes the initialization.

3.3.9 Unindexed event parameters can make offchain tracking less efficient and cumbersome

Severity: Informational

Context: SSS.sol#L68-L84

Description: The `SSS.sol` token contract emits multiple events during critical state changes. However, most event parameters lack the `indexed` keyword, making off-chain tracking less efficient and cumbersome:

```
event SetLiquidityPool(address pool, bool isPool);
event ClaimGasFee(address recipient, uint256 amount);

event DevClaimTax(address to, uint256 amount);
event DevClaimUnlockToken(address to, uint256 amount);
event CommunityClaimTax(address to, uint256 amount);

event MigrationCompleted();
event SetPreMigrationOperator(address operator, bool isOperator);
event SetRestrictedTradingPool(address pool, bool isRestricted);
event SetCommunityAddress(address community);
event SetDevAddress(address devTaxReceiver, address devTokenReceiver);
event ChangeTaxPercent(uint256 buyTax, uint256 sellTax, uint256 dev, uint256 community);
event SetExcludeFromTax(address account, bool exclude);
event RescueToken(address tokenAddress, address to, uint256 amount);
event RescueETH(uint256 amount, bool success);
event InitPool(uint256 ethAmount, uint256 tokenAmount);
```

Recommendation: Consider adding the `indexed` keyword to parameters of events that are frequently queried (or filtered). For example,

```
- event DevClaimTax(address to, uint256 amount);
+ event DevClaimTax(address indexed to, uint256 amount);

- event DevClaimUnlockToken(address to, uint256 amount);
+ event DevClaimUnlockToken(address indexed to, uint256 amount);

- event CommunityClaimTax(address to, uint256 amount);
+ event CommunityClaimTax(address indexed to, uint256 amount);
```

SSS: Addressed in commit d41e6166.

Cantina: Reviewed that this is fixed as recommended for the examples highlighted above.