PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Taco Party (Salsa Edition)

18 November 2021

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

1 Overview

This report has been prepared for Taco Party (Salsa Edition) on the Polygon network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Taco Party (Salsa Edition)
URL	https://salsa.tacoparty.finance/farms
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
SalsaChefV2	0xDe1c7b7126aada7bBaDb28C79c5FB20Ba667b85F	MATCH
PreSalsaToken	0xEa3B6dA2031DF019EE48EE982E284dd4dCBBCc31	MATCH
PreSalsaSwap	0x1E0A8F47e894d5a77f300935a33381164308Ba81	MATCH
SalsaBunnies	0x9C0f8e9796A334c7b95551dbeCDadd43F7d87217	MATCH
SalsaBunniesFarm	0xbbF6eB02755b8c3e9470cc03C5526401Ec0ABC0F	MATCH
SalsaToken	0x64367C7A9e91da86386964DCFfECf0EC48D2fbc0	MATCH
Multicall	Only used by frontend	MATCH
Timelock	0x97A53314041A689956cE66bB74907FcCEbAF001f	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	2	1	-
Medium	2	2	-	-
Low	10	7	1	2
Informational	20	15	-	5
Total	**35**	**26**	**2**	**8**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 SalsaChefV2

ID	Severity	Summary	Status
01	HIGH	The price based emissions can be freely manipulated	PARTIAL
02	MEDIUM	Max supply check does not account for dev reward	RESOLVED
03	LOW	`user.lastWithdraw` is not set on `emergencyWithdraw`	RESOLVED
04	LOW	setEmissionRate is redundant to updateEmissionRate	RESOLVED
05	INFO	`msg.sender` is unnecessarily cast to `address(msg.sender)`	RESOLVED
06	INFO	`usdc` can be made immutable	RESOLVED
07	INFO	`salsaMaximumSupply` can be made constant	RESOLVED
08	INFO	`setEmissionRate`, `setUSDCSalsaLPAddress`, `updateEmissionParameters` and `updateEmissionRate` can be made `external`	RESOLVED
09	INFO	Lack of events for `setUSDCSalsaLPAddress` and `updateEmissionParameters`	RESOLVED

1.3.2 PreSalsaToken

ID	Severity	Summary	Status
10	LOW	Per-user limit of 500 PreSalsa does not apply allowing users to purchase significantly more tokens than the intended limit	ACKNOWLEDGED
11	LOW	`maxHardCap` is out of range	ACKNOWLEDGED
12	INFO	`USDC`, `presaleAddress`, `salePrice`, `maxHardCap`, can be made `constant`	ACKNOWLEDGED
13	INFO	`presaleAddress` is a misnomer	ACKNOWLEDGED
14	INFO	Contract unnecessarily stores its own address in a variable which could confuse third-party reviewers	ACKNOWLEDGED
15	INFO	Contract only works with 6 decimal USDC	ACKNOWLEDGED

1.3.3 PreSalsaSwap

ID	Severity	Summary	Status
16	LOW	`salsaAddress` and `hasBurnedUnsoldPresale` are `private`	RESOLVED
17	INFO	Unused variable: `redeemState`	RESOLVED
18	INFO	Inconsistent usage of SafeERC20	ACKNOWLEDGED

1.3.4 SalsaBunnies

ID	Severity	Summary	Status
19	LOW	Contract wrongly increments the bunny count on burn instead of decrementing it	RESOLVED
20	LOW	Unused ContextMixin	RESOLVED

1.3.5 SalsaBunniesFarm

ID	Severity	Summary	Status
21	LOW	Lack of end block validation on `mintNFT`	RESOLVED
22	INFO	`salsaBunnies`, `salsaToken`, `endBlockNumber`, `salsaPerBurn` and `totalSupplyDistributed` can be made `immutable`	RESOLVED
23	INFO	`numberOfBunnyIds` can be made constant	RESOLVED
24	INFO	Unused variable: `startBlockNumber`	RESOLVED
25	INFO	Lack of events for `setStartBlockNumber`, `whitelistAddresses`, `withdrawSalsa` and `changeOwnershipNFTContract`	RESOLVED

1.3.6 SalsaToken

ID	Severity	Summary	Status
26	HIGH	updateSalsaSwapRouter could be used to break transfer and siphon all transfer taxes	RESOLVED
27	HIGH	Generated liquidity is sent to the operator which could potentially dump it	RESOLVED
28	MEDIUM	Gov privilege: Lack of lower limit on minAmountToLiquify potentially breaking transfers	RESOLVED
29	LOW	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	PARTIAL
30	LOW	`_isExcludedFromFee` is private	RESOLVED
31	INFO	mint, excludeFromFee, includeInFee, setMinAmountToLiquify, updateSwapAndLiquifyEnabled, updateSalsaSwapRouter and transferOperator can be made external	RESOLVED
32	INFO	Lack of events for excludeFromFee and includeInFee	RESOLVED
33	INFO	Usage of require when assert is more appropriate	RESOLVED
34	INFO	Typographical errors	RESOLVED
35	INFO	`msg.sender` is unnecessarily cast to `address(msg.sender)`	RESOLVED

1.3.7 Multicall

No issues found.

1.3.8 Timelock

No issues found.

2 Findings

2.1 SalsaChefV2

The SalsaChefV2 is a highly customized Masterchef based on Goose's masterchef. The team has improved on this Masterchef by implementing many of Paladin's recommendations including the limiting deposit fees to at most 4.01%.

The most notable feature of SalsaChefV2 is that emissions are variable upon the salsa price – they can reach between 1 and 100 tokens per block and will speed up as price goes down. The second notable feature is a bonus for not withdrawing.

2.1.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `add`
- `set`
- `setFeeAddress`
- `setStartBlock`
- `setEmissionRate`
- `setUSDCSalsaLPAddress`
- `updateEmissionParameters`
- `updateEmissionRate`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	The price based emissions can be freely manipulated
Severity	HIGH SEVERITY

Location	Lines 393-397
	`uint salsaBalance = salsa.balanceOf(usdcSalsaLP);`
	
	`if (salsaBalance > 0) {`
	`// usdc token decimals = 6, token decimals = 18 ,(18-x)=12 + 2 = 14 to convert to cents`
	`priceCents = usdc.balanceOf(usdcSalsaLP) * 1e14 / salsaBalance;`

Description	Currently the emission rate is based upon the token price which is calculated from the token balances of the LP pair. However, users can freely deposit tokens into the LP pair and instantly take them out again using the `skim()` method to manipulate this price at zero cost. Furthermore, people can take out tokens to manipulate this price at a cost of 0.3% of the tokens taken out. ❗ Expressing the price in cents could furthermore be impressive if the token is not sufficiently valuable. ❗ The 100 in `(salsaPriceCents * 100) / (topPrice * 100)` is furthermore redundant as it is used on both sides of the division. ❗ The line `salsaPerBlock = MAX_EMISSION_RATE / 100 * emissionRatePercent;` does division before multiplication which is considered a bad pattern
Recommendation	Consider using a TWAP oracle, alternatively, to go to the 0.3% cost, at least `getReserves()` could be considered instead of using raw LP balances. Consider removing the two 100s and consider doing multiplication before division.

Resolution	PARTIALLY RESOLVED `getReserves` is now used and the 100s as mentioned in the recommendation are removed. This issue is marked as partially resolved as some price manipulation is still possible, but it will come with a swapping cost of at least 10.6% the swapped tokens as there is also a 5% transfer-tax. Such an attack, while still possible, would therefore be extremely expensive. A small note for potential forks is also that the logic assumes a USDC token with 6 decimals, which is not the case on all chains.

Issue #02	**Max supply check does not account for dev reward**
Severity	MEDIUM SEVERITY

Location	Lines 227-228
	`else if ((salsa.totalSupply() + salsaReward) > salsaMaximumSupply)`
	`    salsaReward = salsaMaximumSupply - salsa.totalSupply();`

Description	The maximum supply check does not account for the fact that 10% tokens are minted extra to the `feeAddress`. This causes the maximum supply to be exceeded.
Recommendation	Consider accounting for the dev mint and simply setting it to zero in this scenario (the amount also needs to be included in the if statement). Furthermore consider removing the excessive brackets within the `if` statement.
Resolution	RESOLVED The dev reward is now accounted for.

Issue #03	`user.lastWithdraw` is not set on `emergencyWithdraw`
Severity	LOW SEVERITY
Description	Currently the `lastWithdraw` parameter is not set on `emergencyWithdraw`. `lastWithdraw` is furthermore not set on `deposit` which might not be the intention of this logic. If it is not set during the first deposit, `blockDifference` is a very large number. ❗ Another futile multiplication and division by 100 is made within `getStakingRewardsMultiplier`. Consider reorganizing this to do multiplication before division and this can be removed.
Recommendation	Consider setting `user.lastWithdraw` on `emergencyWithdraw`.
Resolution	RESOLVED This is now set to the latest block on emergency withdraw. It is now furthermore set on the first deposit.

Issue #04	`setEmissionRate` is redundant to `updateEmissionRate`
Severity	LOW SEVERITY
Description	Currently the code contains two functions with identical behavior. This is unnecessary and could annoy users that might want to listen to the emitted events.
Recommendation	Consider removing either of these functions.
Resolution	RESOLVED `updateEmissionRate` has been removed.

Issue #05	`msg.sender` is unnecessarily cast to `address(msg.sender)`
Severity	INFORMATIONAL
Description	`msg.sender` is cast to `address(msg.sender)` throughout the contract when used with `pool.lpToken.safeTransfer()`. This is unnecessary.
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with `msg.sender`.
Resolution	RESOLVED

Issue #06	`usdc` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the the variable explicitly `immutable`.
Resolution	RESOLVED

Issue #07	`salsaMaximumSupply` can be made `constant`
Severity	INFORMATIONAL
Description	Variables that are never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the aforementioned variables explicitly constant.
Resolution	RESOLVED

Issue #08	**`setEmissionRate`, `setUSDCSalsaLPAddress`, `updateEmissionParameters` and `updateEmissionRate` can be made `external`**
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above variables as external.
Resolution	RESOLVED

Issue #09	**Lack of events for `setUSDCSalsaLPAddress` and `updateEmissionParameters`**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

2.2 PreSalsaToken

The PreSalsaToken is a simple presale contract that allows users to buy presale Salsa with USDC at a fixed price.

30,000 tokens are mint to for the presale while 5,000 tokens are mint to governance (0xCf7Db495dFb74302870fFE4aC8D8d19550d97fA8). For each 100 USD provided, a user will receive 25 PreSalsaToken.

2.2.1 Privileged Roles

The following functions can be called by the owner of the Masterchef:

- `setStartBlock`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #10	**Per-user limit of 500 PreSalsa does not apply allowing users to purchase significantly more tokens than the intended limit**
Severity	LOW SEVERITY
Location	Line 73 `require(userPreSalsaTotally[msg.sender] < maxPreSalsaPurchase, "user has already purchased too much presalsa");`
Description	Currently the limit of 500 PreSalsaToken is enforced before the purchase has been made. However, this means that if someone were to purchase 30,000 tokens at once, this would pass as their initial balance is less than 500 tokens.
Recommendation	Consider moving the requirement to line 76 and adjusting it to the following requirement: `require(userPreSalsaTotally[msg.sender] + presalsaPurchaseAmount <= maxPreSalsaPurchase, "user has already purchased too much presalsa");`
Resolution	ACKNOWLEDGED This contract has already been deployed and the client has indicated they will take this behavior into consideration.

Issue #11	`maxHardCap` is out of range
Severity	LOW SEVERITY
Description	Currently the contract contains a limit to the amount of USDC that can be sold in a single transaction (note that this is not a total in the current implementation). As this limit is set at $150k USDC and the presale is for $120k at the current price, this variable is out of range. ❗ If this is supposed to be a global limit, consider adjusting the business logic as it currently applies on a per-transaction level.
Recommendation	Consider removing the `maxHardCap` variable or putting it to a variable within range.
Resolution	ACKNOWLEDGED This contract has already been deployed and the client has indicated they will take this behavior into consideration.

Issue #12	`USDC`, `presaleAddress`, `salePrice` and `maxHardCap` can be made `constant`
Severity	INFORMATIONAL
Description	Variables that are never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly constant.
Resolution	ACKNOWLEDGED

Issue #13	**presaleAddress is a misnomer**
Severity	INFORMATIONAL
Location	Line 30 `address constant presaleAddress = 0xCf7Db495dFb74302870fFE4aC8D8d19550d97fA8;`
Description	The contract calls the governance wallet `presaleAddress` which is misleading to third-party validators. ❗ The contract also contains a comment with "Ratio 1:0.993" while the swapping ratio has a 1:1 ratio.
Recommendation	Consider renaming this variable to `feeAddress` or similar. Consider removing the obsolete comment. Note that SalsaToken a transfer tax so if the swap is included the ratio would be 1:0.95
Resolution	ACKNOWLEDGED

Issue #14	**Contract unnecessarily stores its own address in a variable which could confuse third-party reviewers**
Severity	INFORMATIONAL
Location	Line 34 `IERC20 preSalsaToken = IERC20(address(this));`
Description	The `preSalsaToken` is unnecessary. It makes it appear throughout the code that external contract calls are made while in fact they are not.
Recommendation	Consider simply using `address(this)` or calling `transfer` wherever `preSalsaToken` is currently used. As we know the implementation of this contract, SafeERC20 is not required.
Resolution	ACKNOWLEDGED

Issue #15	Contract only works with 6 decimal USDC
Severity	INFORMATIONAL
Description	The contract currently only works with an USDC contract with 6 decimals. As not all USDC contracts are 6 decimals the user has to be careful with redeploying this contract on other chains. However, within Polygon, the logic is correct.
Recommendation	Consider either using the `.decimals()` value of the token, or carefully adjusting the logic of this contract whenever a new chain is picked.
Resolution	ACKNOWLEDGED

2.3 PreSalsaSwap

The PreSalsaSwap contract allows for the swapping of PreSalsaToken to SalsaToken. This functionality requires that there are sufficient SalsaToken in PreSalsaSwap. If the swap contract is not excluded from the transfer tax, a rate of 1:0.95 would be applied due to the tax, otherwise the rate is 1:1.

2.3.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `sendUnclaimedSalsaToDeadAddress`
- `setStartBlock`
- `transferOwnership`
- `renounceOwnership`

2.3.2 Issues & Recommendations

Issue #16	`salsaAddress` and `hasBurnedUnsoldPresale` are `private`
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as `public` so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the above variables as public.
Resolution	RESOLVED

Issue #17	Unused variable: `redeemState`
Severity	INFORMATIONAL
Description	Variables defined in a contract but not used within said contract could confuse third-party auditors. They furthermore increase the contract length and bytecode size for no reason.
Recommendation	Consider removing the variables to keep the contract short and simple.
Resolution	RESOLVED

Issue #18	Inconsistent usage of SafeERC20
Severity	INFORMATIONAL
Location	Line 62 `require(preSalsaToken.transferFrom(msg.sender, BURN_ADDRESS, swapAmount), "failed sending presalsa" );`
Description	Throughout the contracts SafeERC20 is used except on line 62.
Recommendation	Consider using `safeTransferFrom` on line 62 to be consistent. Note that as we know the implementation of PreSalsaToken this does not make a difference with regards to contract functionality.
Resolution	ACKNOWLEDGED

2.4 SalsaBunnies

The SalsaBunnies contract is a simple NFT contract which allows the Salsa team to define bunnies – for each bunny type (with a unique name), multiple unique NFTs can be minted.

Pre-approval to OpenSea is given, according to the OpenSea recommendations.

2.4.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `mint`
- `setBunnyName`
- `burn`
- `transferOwnership`
- `renounceOwnership`

2.4.2 Issues & Recommendations

Issue #19	Contract wrongly increments the bunny count on burn instead of decrementing it
Severity	LOW SEVERITY
Location	Line 119 `bunnyCount[bunnyIdBurnt] += 1;`
Description	The contract increments instead of decrementing the bunny count. As this count is used for UI purposes, it should not affect users.
Recommendation	Consider adjusting the previous line to: `bunnyCount[bunnyIdBurnt] -= 1;`
Resolution	RESOLVED The variable is now decremented.

Issue #20	Unused ContextMixin
Severity	LOW SEVERITY
Location	Line 36 `contract SalsaBunnies is ERC721URIStorage, ContextMixin, Ownable {`
Description	The contract implements a simple meta transaction scheme to allow third parties to execute transactions as if a user executed them. However, this feature is not actively used as there is no function to actually create meta transactions. This issue is marked as low severity instead of informational as ContextMixin is a scary dependency for some third-party reviewers given that it is low-level.
Recommendation	Consider removing the ContextMixin dependency, otherwise consider implementing it fully.
Resolution	RESOLVED The ContextMixin dependency has been removed.

2.5 SalsaBunniesFarm

The SalsaBunniesFarm is the contract where users can mint SalsaBunnies from, it should therefore be the owner of SalsaBunnies which it initially deploys and is therefore owner. Users can choose their bunny id freely all though it must be either 0, 1 or 2.

The bunny options are:
- Rhythm of Passion
- Buy Salsa
- Salsa Goodness

Wallets must be whitelisted and can only claim up to 1 of these three bunnies once. Only up to a configurable amount of bunnies can be purchased in a first-come-first-served manner among the whitelisted addresses. Users must have some balance of salsa, however small, at the time of purchase.

Eventually, if users desire, they can burn their Bunny in this contract to receive an amount of salsa which is configurable by the governance. This can only be done until the end block has been reached.

Ownership of SalsaBunnies can finally be reclaimed by the governance to potentially mint more bunnies for themselves.

2.5.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `setStartBlockNumber`
- `whitelistAddresses`
- `withdrawSalsa`
- `changeOwnershipNFTContract`
- `transferOwnership`
- `renounceOwnership`

2.5.2 Issues & Recommendations

Issue #21	Lack of end block validation on `mintNFT`
Severity	LOW SEVERITY
Description	Currently `mintNFT` does not validate the time of minting, therefore bunnies can be minted indefinitely. ❗ `burnNFT` furthermore says that there is a cap on the amount of bunnies but there is no cap within `burnNFT`.
Recommendation	Consider whether minting should finish once the end block has been reached and then consider enforcing this.
Resolution	RESOLVED There is now end block validation within the `mintNFT` function.

Issue #22	`salsaBunnies`, `salsaToken`, `endBlockNumber`, `salsaPerBurn` and `totalSupplyDistributed` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly immutable.
Resolution	RESOLVED Most of the variables have been made immutable.

Issue #23	**numberOfBunnyIds can be made constant**
Severity	INFORMATIONAL
Description	Variables that are never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly constant.
Resolution	RESOLVED

Issue #24	**Unused variable: `startBlockNumber`**
Severity	INFORMATIONAL
Description	Variables defined in a contract but not used within said contract could confuse third-party auditors. They furthermore increase the contract length and bytecode size for no reason.
Recommendation	Consider removing the variable to keep the contract short and simple.
Resolution	RESOLVED

Issue #25	**Lack of events for `setStartBlockNumber`, `whitelistAddresses`, `withdrawSalsa` and `changeOwnershipNFTContract`**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED `setStartBlockNumber` has been removed and the other functions have received events.

2.6 SalsaToken

The Salsa token is an ERC-20 token extended with a 5% transfer tax which is entirely used for liquidity generation. During contract creation, 50,000 tokens are minted to the deployer. After this, the deployer can mint more tokens until ownership is transferred to the Masterchef.

Ownership of the token should be transferred to the Masterchef which will then continuously mint it for native rewards. The token also incorporates the EIP-2612 `permit` scheme, which allows for transactionless approvals as is famously known with breaking up Uniswap LP tokens.

2.6.1 Token Overview

Address	TBD
Token Supply	None
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	50,000

2.6.2 Privileged Roles

The following functions can be called by the owner of the contract:

- `mint`

The following functions can be called by the operator of the contract:

- `excludeFromFee`
- `includeInFee`
- `setMinAmountToLiquify`
- `updateSwapAndLiquifyEnabled`
- `updateSalsaSwapPair`
- `transferOperator`

2.6.3 Issues & Recommendations

Issue #26	**`updateSalsaSwapRouter` could be used to break transfer and siphon all transfer taxes**
Severity	HIGH SEVERITY
Location	Line 266 `function updateSalsaSwapRouter(address _router) public onlyOperator {`
Description	The contract allows the change of the Uniswap router which is responsible for the liquidity generation mechanism. If this contract is however changed to a malicious one, this could allow the governance to steal all transfer taxes or worse break all transfers.
Recommendation	Consider removing this function. It is very rare that any protocol should ever require changing the router, especially if they have already hard-coded and use an established, prominent router contract.
Resolution	RESOLVED `updateSalsaSwapRouter` has been removed.

Issue #27	**Generated liquidity is sent to the operator which could potentially dump it**
Severity	HIGH SEVERITY
Location	Line 248 `operator(),`
Description	All of the liquidity generated by the transfer-tax is sent to the operator, which is likely to be an EOA. This operator could potentially dump this at some point.
Recommendation	Consider either sending the liquidity directly to the burn address, or to a locking contract.
Resolution	RESOLVED The generated liquidity is now burned.

Issue #28	**Gov privilege: Lack of lower limit on `minAmountToLiquify` potentially breaking transfers**
Severity	MEDIUM SEVERITY
Description	Currently the `minAmountToLiquify` parameter can be set to zero. If this is done, the contract will attempt generating liquidity even if it has no tokens to do so. Due to the uniswap contracts their implementation, this operation will revert. This would break all normal transfers.
Recommendation	Consider adding a lower limit to `minAmountToLiquify`: `require(amount >= MIN_AMOUNT_LIQUIFY_MIN, 'liq min too low);` In addition, consider wrapping all Uniswap operations (`swap` and `addLiquidity`) in an `if` statement that only executes the Uniswap operation if both amounts are larger than zero for `addLiquidity` and the swap if the input amount is greater than zero.
Resolution	RESOLVED The client has added the safeguard.

Issue #29	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef
Severity	LOW SEVERITY
Description	The `mint` function could be used to pre-mint tokens for legitimate uses including, but not limited to, the injection of initial liquidity, token presale, or airdrops; however, this function may also be used to pre-mint and dump tokens when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this `mint` function is to be used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	PARTIALLY RESOLVED The client has explained that disclosing this has been a standard practice for them and is already done here: https://tacosupreme.gitbook.io/taco-party-salsa/taconomics This issue will be fully resolved once ownership is transferred to the Masterchef.

Issue #30	`_isExcludedFromFee` is private
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the variable as public.
Resolution	RESOLVED

Issue #31	`mint`, `excludeFromFee`, `includeInFee`, `setMinAmountToLiquify`, `updateSwapAndLiquifyEnabled`, `updateSalsaSwapRouter` and `transferOperator` can be made `external`
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above variables as external.
Resolution	RESOLVED

Issue #32	Lack of events for `excludeFromFee` and `includeInFee`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

Issue #33	Usage of require when assert is more appropriate
Severity	INFORMATIONAL
Location	Line 160 `require(amount == sendAmount + taxAmount, "tax value invalid");`
Description	Requirements which should never fail are better marked with the assert keyword. This signals to both the compiler and the third-party reviewers that this event can never occur.
Recommendation	Consider using `assert` instead.
Resolution	RESOLVED

Issue #34	Typographical errors
Severity	INFORMATIONAL
Location	Line 89 `require(feeaddr != address(0x0), 'usdc is zero');`
Description	The contract contains the following typographical errors: This error should say fee address instead of usdc.
Recommendation	Add events for the aforementioned functions.
Resolution	RESOLVED

Issue #35	`msg.sender` is unnecessarily cast to `address(msg.sender)`
Severity	INFORMATIONAL
Description	The msg.sender is cast to address(msg.sender) when used with `mint`. This is unnecessary.
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with `msg.sender`.
Resolution	RESOLVED

2.7 Multicall

The Multicall contract is a perfect replica of the Maker DAO Multicall V1 contract. This contract is used solely for frontend purposes to fetch many values from multiple contracts in one RPC call. This allows the frontend to be more responsive. It should be noted that if a single subcall fails, all subcalls fail. This behavior has changed in Multicall V2 which could be considered if desired.

2.7.1 Issues & Recommendations

No issues found.

2.8 Timelock

The Timelock contract is a clean fork of Compound Finance's timelock. This is the most common contract used in DeFi to time lock governance access and is thus compatible with most third-party tools.

Parameter	Value	Description
Delay	6 hours	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	4 hours	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

2.8.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY