PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For RobustSwap

23 October 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 RobustSwap by Robust Protocol on the Binance Smart Chain (BSC). 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	Robust Swap by Robust Protocol
URL	https://robustprotocol.fi/
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`Token`	`0x95336aC5f7E840e7716781313e1607F7C9D6BE25`	MATCH
`MasterChef`	`0xE40b415C28eC411Cc616ca04a125d7e2b9913b58`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	4	3	1	-
Medium	3	2	1	-
Low	2	2	-	-
Informational	10	9	-	1
Total	**19**	**16**	**2**	**1**

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 RobustSwap Token

ID	Severity	Summary	Status
01	HIGH	DoS: An exploiter could potentially trigger the botguards on trading pairs, other wallets or the router to prevent certain swap operations from occurring	RESOLVED
02	HIGH	The pending operator can reclaim ownership after ownership has been transferred to the timelocked operator	RESOLVED
03	HIGH	Governance privilege: Governance can update the router to potentially steal transfer taxes or turn the token into a honeypot	PARTIAL
04	MEDIUM	Governance privilege: The governance wallet can withdraw generated liquidity and any other tokens and BNB in the token contract	PARTIAL
05	MEDIUM	Privilege escalation: An EOA operator can be set as the timelock operator to run timelocked transactions	RESOLVED
06	MEDIUM	An exploiter can avoid the transfer tax, anti-bot guard and trading enabled check by strategically interacting with the swap router	RESOLVED
07	LOW	`addLiquidity` logic is not precise	RESOLVED
08	INFO	Typographical errors and inconsistent formatting	RESOLVED
09	INFO	EnumerableSets are never enumerated	RESOLVED
10	INFO	Gas optimization: Usage of smaller types than uint256 does not save gas	ACKNOWLEDGED
11	INFO	Gas optimization: `minAutoTriggerAmount` can be cached in `autoLiquidity`	RESOLVED
12	INFO	Emitting an event in the fallback function is bad practice	RESOLVED
13	INFO	`transferTaxEnabled` cannot be disabled by governance	RESOLVED

1.3.2 MasterChef

ID	Severity	Summary	Status
14	HIGH	Deposit fees do not have an upper limit	RESOLVED
15	LOW	`updateEmissionRate` has no maximum safeguard	RESOLVED
16	INFO	`BONUS_MULTIPLIER` is redundant	RESOLVED
17	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED
18	INFO	Accounting for maximum token supply and minting can be improved	RESOLVED
19	INFO	There are no sanity checks on the `setRBSReferralContract`	RESOLVED

2 Findings

2.1 RobustSwap Token

The RobustSwap token is a customized deflationary ERC-20 token with transfer taxes used for liquidity generation and burning, an anti-bot cooldown period on trades and a limited mintable supply. A maximum amount of 106,050 tokens can be minted, and 930 tokens will be pre-minted for various purposes including airdrops and liquidity.

By default, the contract and the contract creator are excluded from the transfer limit and transfer tax.

The total transfer tax can be set to a maximum of 20% and can be adjusted for purchases and sales. This tax will be partially burned and partially used for liquidity generation. This split can be freely determined by the governance. The initial taxes are set at 6% for purchases and 8% for taxes. Of this transfer tax, initially 0% is burned which means it is used completely for liquidity generation. Generated liquidity is sent to the token contract and not burned.

A transfer limit is imposed on transactions but cannot be set to a value lower than 0.1%. The initial transfer limit is set to 1% of the total minted supply. The maximum cool down period for bots is 10 blocks, which means they can only make a purchase or sale every 10 blocks. It is initially set to 5 blocks.

2.1.1 Privileged Roles

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

- `mint`
- `transferOwnership`
- `renounceOwnership`
- `enableTrading`
- `updateRateTax`
- `updateRateBurn`
- `updateRateTransferLimit`
- `updateTransferLimitExclusionAdd`
- `updateTransferLimitExclusionRemove`
- `updateTransferTaxEnabled`
- `updateTaxExclusionAdd`
- `updateTaxExclusionRemove`
- `updatePairListAdd`
- `updatePairListRemove`
- `updateAutoTrigger`
- `updateBotGuard`
- `updateAutoSellEnabled`
- `updateAutoLiquidityStatus`
- `updateRobustSwapRouter`
- `balanceBurnRBS`
- `balanceWithdrawToken`
- `balanceWithdrawBNB`
- `updateOperatorSetTimeLock`
- `updateOperatorSetPending`
- `updateOperatorAcceptPending`

2.1.2 Issues & Recommendations

Issue #01	**DoS: An exploiter could potentially trigger the botguards on trading pairs, other wallets or the router to prevent certain swap operations from occurring**
Severity	HIGH SEVERITY
Description	The botguard will be enabled on the receipt of tokens when a purchase transaction is detected; however, malicious parties can purchase tokens for any wallet of their choosing enabling the bot guard on these targeted wallets. After the guard has been enabled, these wallets cannot make any transfers for a few blocks. This could allow malicious parties to front-run large sales and prevent them, and it could also be used to trigger the guard on a very high-valued Uniswap pair (not the main one) which would cause the frontend to try to sell on this pair while these sales will revert. This would turn the token effectively into a honeypot. Finally, this could be abused to trigger the anti-bot on the router as well. ❗ We also do not see any reason for having `sender == routerAddress()` and `recipient == routerAddress()` in the `taxedTransfers` method as purchases and sales occur directly from the user to the pair.
Recommendation	Consider removing the anti-bot mechanism. Otherwise, consider just adding the bot guard on `tx.origin`, the wallet that instantiated the transaction. This is imperfect as well but likely any anti-bot mechanism can be circumvented. At least this method does not have side-effects.
Resolution	RESOLVED The client has stated that this is a desired feature, and has implemented the recommendation to enforce `botGuard` on `tx.origin`. Additionally, `routerAddress` has been removed from both buy and sell transactions.

Issue #02	**The pending operator can reclaim ownership after ownership has been transferred to the timelocked operator**
Severity	HIGH SEVERITY
Description	The code contains a function to explicitly transfer ownership to a timelocked operator. Although this function is not much different from the other operator transfership function, it does not override the `pendingOperator` causing this pending operator, if one exists, to be able to reclaim the ownership after this transfer is done, undoing a potential timelock safeguard.
Recommendation	Consider also setting the `pendingOperator` to the zero address within the function that transfers the operatorship to the "timelocked operator".
Resolution	RESOLVED Resolved in Issue #05. Note that the `operatorPending` variable can thus be removed.

Issue #03	**Governance privilege: Governance can update the router to potentially steal transfer taxes or turn the token into a honeypot**
Severity	HIGH SEVERITY
Location	Line 2277 `function updateRobustSwapRouter(address _routerAddress) external onlyOperator {`
Description	The governance can change the Uniswap router which is used for liquidity generation to any contract of their choosing. In case this contract is set to a malicious contract created by the governance, it could steal transfer taxes or revert transactions to only allow purchases to pass, turning the token into a honeypot.
Recommendation	Consider removing the `updateRobustSwapRouter` function.
Resolution	PARTIALLY RESOLVED The client has stated that they will be launching their own AMM in the future and thus require this function. Unfortunately, we are unable to mark this issue as Resolved as the risks of this issue remain present. However, given that the Robust Protocol is relatively established, the probability of this function being used to update to a malicious Router is low.

Issue #04	**Governance privilege: The governance wallet can withdraw generated liquidity and any other tokens and BNB in the token contract**
Severity	MEDIUM SEVERITY
Description	The governance can withdraw the generated liquidity LP tokens from the contract to potentially dump them. Furthermore, BNB and native tokens that are still in the contract from the liquidity generation mechanism can be withdrawn by the governance as well. This might be used if they notice the project is failing and they want to exit profitably.
Recommendation	Consider sending the generated LP to an explicit locking contract or excluding them to be withdrawn before a certain date. In addition, consider addressing the potential that BNB and the native token is withdrawn maliciously by either having a good governance structure, removing these functions, or only making them callable after a certain timestamp has passed.
Resolution	PARTIALLY RESOLVED The `balanceWithdrawBNB` function has been removed, but the ability to withdraw LP and all other tokens (excluding native) still remains. The client has stated that this ability is required for future migration purposes. This function can only be called when behind the set timelock.

Issue #05	**Privilege escalation: An EOA operator can be set as the timelock operator to run timelocked transactions**
Severity	MEDIUM SEVERITY

Location	Lines 2009-2018
	`function updateOperatorSetTimeLock(address _timeLockContract) external onlyOperator {`
	`  require(_timeLockContract != address(0), "RobustSwap::updateOperatorSetTimeLock:Timelock cannot be the zero address");`
	`  require(_timeLockContract != operator, "RobustSwap::updateOperatorSetTimeLock:Current operator address cannot be used");`
	
	`  opeatorTimeLocked = true;`
	
	`  emit UpdateOperatorSetTimeLock(operator, _timeLockContract, opeatorTimeLocked);`
	
	`  operator = _timeLockContract;`
	`}`

Description	The code allows the operator to escalate their privileges to allow them to call timelocked operations since there is no validation done that `operator` is in fact a timelock contract after `updateOperatorSetTimelock` is called.
Recommendation	Consider acknowledging the fact that there is no easy way to validate that an address is in fact a proper timelock contract. Instead the `timelockedOperator` modifier can be removed since it provides little security in favor of actually just timelocking the operator or adding a second `timelockedOperator` address to control the more sensitive functions. This second operator would be a timelock from the start.
Resolution	RESOLVED The Timelock contract will be initialized in the constructor, and the `updateOperatorSetTimeLock` function will allow for the operator to be updated to this Timelock contract only once, with the ability to update the `operatorTimelock` contract removed.

Issue #06	**An exploiter can avoid the transfer tax, anti-bot guard and trading enabled check by strategically interacting with the swap router**
Severity	MEDIUM SEVERITY
Location	Line 1658 `bool isLiquidityTransfer = ((isRobustSwapPair(sender) && recipient == routerAddress()) || (isRobustSwapPair(recipient) && sender == routerAddress()));`

Description	The project attempted to make LP addition and removal exempt from the transfer tax. However, due to this attempt, many functionalities can be strategically avoided if we can somehow mimic that we are doing an LP addition or removal, while we are in-fact making a purchase or sale.
	
	Such a method is in fact possible, for example through the `removeLiquidityETHSupportingFeeOnTransferTokens` function on the router:
	
	`function removeLiquidityETHSupportingFeeOnTransferTokens(`
	`    ...`
	`) public virtual override ensure(deadline) returns (uint amountETH) {`
	`    (, amountETH) = removeLiquidity(`
	`        ...`
	`    );`
	`    TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(address(this)));`
	`    IWETH(WETH).withdraw(amountETH);`
	`    TransferHelper.safeTransferETH(to, amountETH);`
	`}`
	
	This function transfers the complete balance of the router to the `to` address. An exploiter could thus for example sell tokens without taxes and without the trading enabled check or anti-bot guard triggering by setting the address to the pair and then withdrawing WBNB from the pair directly.
	
	It should be noted that this issue is marked as medium and not high since there might not be easy ways for the exploiter to get the native tokens to and from the router while swapping has not been enabled. However, once swapping is enabled, this method should allow both selling and purchasing of the native token without transfer taxes.

Recommendation	Consider not excluding the swap router from the tax behavior.

Resolution	RESOLVED The swap router is no longer excluded from the tax behaviour.

Issue #07	**`addLiquidity` logic is not precise**
Severity	LOW SEVERITY
Description	The add liquidity logic does not account for slippage and transfer taxes when the token value is calculated, and `amountRBSPerBNB` is not precise since Solidity does not handle decimals and rounding errors could occur if RBS ever becomes valuable relative to BNB.
Recommendation	Consider making `amountRBSPerBNB` more precise by using it directly in the multiplication below it. Furthermore, consider the impact of the rounding errors of this variable. ❗ Governance should likely also limit the `autoTriggerRate` to less than 50%.
Resolution	RESOLVED The client has implemented rigorous testing and there are no issues present with the current logic.

Issue #08	Typographical errors and inconsistent formatting
Severity	INFORMATIONAL

Description	The code is formatted using both tabs and spaces which makes the code look very odd on certain IDEs. In addition, the code contains a few typographical errors.
	
	Line 1437
	`bool public opeatorTimeLocked = false;`
	
	This should be `operatorTimeLocked`.
	
	Line 1737
	`// Buy Transaction`
	
	This should in fact say "Sell Transaction".

Recommendation	Consider fixing the above typographical errors.
Resolution	RESOLVED

Issue #09	EnumerableSets are never enumerated
Severity	INFORMATIONAL
Description	The code uses the EnumerableSet dependency on various locations. Although it is a useful dependency, it should be avoided when not necessary since it makes third-party reviewing more intensive. In this case the enumerable properties are not used and a mapping could as well be used.
Recommendation	Consider using `mapping(address=> bool)` instead of enumerable sets to reduce the code complexity and peer-review cost.
Resolution	RESOLVED

Issue #10	**Gas optimization: Usage of smaller types than `uint256` does not save gas**
Severity	INFORMATIONAL
Description	The code uses smaller unsigned integers for various variables. However, since word space within Solidity is actually 256 bits, this does not save gas. Instead, it consumes slightly more gas since translations are necessary.
Recommendation	Consider always using `uint256`.
Resolution	ACKNOWLEDGED

Issue #11	**Gas optimization: `minAutoTriggerAmount` can be cached in `autoLiquidity`**
Severity	INFORMATIONAL
Description	The code recomputes the minimum auto trigger amount three which causes extra gas to be used unnecessarily.
Recommendation	Consider storing the result of `minAutoTriggerAmount` ones and using that cached variable three times.
Resolution	RESOLVED

Issue #12	Emitting an event in the fallback function is bad practice
Severity	INFORMATIONAL
Description	It is generally considered bad practice to emit an event in the fallback transaction since .transfer calls already have a very limited gas stipend. When these calls are done on a fallback function with an even, this stipend becomes even smaller. In case gas usage of transactions is ever tweaked in hard-forks, this could make it impossible for the contract to receive ETH from `.transfer` calls, this is for example done when WBNB is unwrapped.
Recommendation	Consider removing the event.
Resolution	RESOLVED

Issue #13	`transferTaxEnabled` cannot be disabled by governance
Severity	INFORMATIONAL
Description	If governance ever sets the `transferTaxEnabled` variable to `false`, this will be set to `true` again because of the `noTransferTax` modifier which is called during the transfers.
Recommendation	Consider modifying the `noTransferTax` to return to whatever the `transferTaxEnabled` state was before it.
Resolution	RESOLVED

2.2 MasterChef

The RobustSwap Masterchef is a modified fork of the Panther Masterchef. Just like Panther, rewards can only be harvested after a specified interval (configurable to at most 14 days) has passed.

The referral commission rate is initially set to 1% and can be set to a maximum of 10%. Deposit fees are currently uncapped, and the protocol has added an anti-bot feature which imposes a 10 second delay between deposits or withdrawals. Deposits and harvesting are possible after `startBlock` has passed, which means users attempting to pre-stake is not possible.

2.2.1 Privileged Roles

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

- `updateStartBlock`
- `add`
- `set`
- `setDevAddress`
- `setFeeAddress`
- `updateEmissionRate`
- `setRBSReferralContract`
- `updateBotGuard`
- `setReferralCommissionRate`

2.2.2 Issues & Recommendations

Issue #14	Deposit fees do not have an upper limit
Severity	HIGH SEVERITY
Description	Deposit fees can be set to any amount up to 100%, which can result in significant or total loss of funds on unsuspecting users.

Recommendation	It is recommended to set a limit to a more reasonable amount, such as 4% (Currently the norm) or up to 10%, in both add and set functions.
	
	For `add`:
	
	`require(_depositFeeBP <= 400, "add: invalid deposit fee basis points");`
	
	For `set`:
	
	`require(_depositFeeBP <= 400, "set: invalid deposit fee basis points");`

Resolution	RESOLVED Deposit fees are now capped at 10%.

Issue #15	`updateEmissionRate` has no maximum safeguard
Severity	LOW SEVERITY
Description	Projects sometimes accidentally update their emission rate to a severely high number either by accident or with malicious intent.
Recommendation	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_rbsPerBlock <= MAX_EMISSION_RATE,"Too high");`
Resolution	RESOLVED Emission rate is now capped at 0.05 tokens per block.

Issue #16	`BONUS_MULTIPLIER` is redundant
Severity	INFORMATIONAL
Description	The `BONUS_MULTIPLIER` does not look to be used anywhere in the contract, and can thus be removed to reduce the length of the contract.
Recommendation	Consider removing this variable.
Resolution	RESOLVED

Issue #17	**Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `updatePool`, `accRBSPerShare` is based upon the `lpSupply` variable. `pool.accRBSPerShare = pool.accRBSPerShare.add(rbsReward.mul(1e12).div(pool.lpSupply));` However, if this `lpSupply` becomes a severely large value this will cause precision errors due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals.
Recommendation	Consider increasing precision to `1e18` across the entire contract.
Resolution	RESOLVED

Issue #18	Accounting for maximum token supply and minting can be improved
Severity	INFORMATIONAL

Location	Lines 2790-2800
	`uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);`
	`uint256 rbsReward = multiplier.mul(rbsPerBlock).mul(pool.allocPoint).div(totalAllocPoint);`
	
	`//RBS maximum supply minted`
	`if (rbsReward > mintableRBS()) {`
	`  rbsReward = 0;`
	`  pool.allocPoint = 0;`
	`  pool.lastRewardBlock = block.number;`
	`  rbsPerBlock = 0;`
	`  return;`
	`}`

Description	In the `updatePool` function, emissions are halted when `rbsReward` exceeds the remaining RBS to be minted. When that happens, `rbsReward`, `rbsPerBlock` and the pool's `allocPoints` are set to zero. This would accurately stop `pool.accRBSPerShare`, but it would also cause the pool to be withdrawn from the active pool list, and users would not be able to interact on the front-end to withdraw their tokens. Additionally, if the difference between `block.number` and `pool.lastRewardBlock` is rather large, then `rbsReward` may be slightly inflated and thus exceed `mintableRBS`, which would result in the maximum token supply potentially never being reached when token amount is very close to the max supply.

Recommendation	Consider revamping the `updatePool` function so as to not mint tokens in case the supply exceeds the total supply. The least intrusive solution is to change to:
	
	`if (rbs.totalSupply().add(rbsReward.mul(11).div(10)) <= rbs.MAXIMUM_SUPPLY) {`
	`    // The whole emission can be mint`
	`    rbs.mint(devAddress, rbsReward.div(10));`
	`    rbs.mint(address(this), rbsReward);`
	`} else if (rbs.totalSupply() < rbs.MAXIMUM_SUPPLY) {`
	`    // The emission can be partially mint`
	`    rbs.mint(address(this), rbs.MAXIMUM_SUPPLY.sub(rbs.totalSupply());`
	`}`
	
	This will only ever mint the total supply at most.
	
	Note that this modification should also be made in the referral minting code section.
	
	A shorter but more advanced approach could be to simply wrap all mint statements in try/catch structures. Even if the mint fails, the main transaction will still succeed.

Resolution	RESOLVED

Issue #19	**There are no sanity checks on the `setRBSReferralContract`**
Severity	INFORMATIONAL
Description	A lot of functionality can break if the referral address is updated to a value that is not a referral contract.
Recommendation	Consider making the referral address non upgradeable (only settable once) to ensure that functionality can never break. We rarely ever see a project updating their referral after it is initially set.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY