PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For ShivaToken

19 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 ShivaToken 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	ShivaToken
URL	https://shivatoken.club/
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
SHIVA	SHIVA.sol	
SHIVADividendTracker	SHIVADividendTracker.sol	
DividendPayingToken	DividendPayingToken.sol	
IterableMapping	IterableMapping.sol	
Source	https://github.com/ShivaToken/ShivaToken/blob/31714a8d4475e035c3bb660b84a714ba7937a4be/ShivaToken.sol	

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	8	7	-	1
● Medium	4	3	-	1
● Low	4	1	-	3
● Informational	14	10	1	3
Total	**30**	**21**	**1**	**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 SHIVA

ID	Severity	Summary	Status
01	HIGH	Typographical errors cause contract to fail compilation	✓ RESOLVED
02	HIGH	DoS: Tokens can be sent to a user periodically to trigger their swap limit and prevent them from making any transactions	ACKNOWLEDGED
03	HIGH	Gov Privilege: `updateUniswapV2Router` could be used to revert transactions, siphon fees or turn the token into a honeypot	✓ RESOLVED
04	HIGH	Gov privilege: Maximum buy and sell amounts can be set freely to potentially turn the token into a honeypot	✓ RESOLVED
05	HIGH	Gov privilege: Governance can blacklist wallets preventing them from making any further transactions like selling their tokens	✓ RESOLVED
06	HIGH	Maximum buy and sell amount can only be set to infinitesimally small amounts	✓ RESOLVED
07	HIGH	Gov Privilege: Fees are freely adjustable up to over 100%	✓ RESOLVED
08	MEDIUM	Gov Privilege: Owner can update the dividend tracker to siphon all dividends and potentially block sell transactions	✓ RESOLVED
09	MEDIUM	Gov privilege: Maximum anti-whale transfer amount can be set as low as 0.0001% of the total supply, practically disabling all transfers	✓ RESOLVED
10	MEDIUM	Gov privilege: If the marketing wallet is a contract, it could reject the BNB transfers turning the token into a honeypot	ACKNOWLEDGED
11	MEDIUM	`limitSwap` functionality is broken	✓ RESOLVED
12	LOW	Gov privilege: Governance can exclude wallets from receiving dividends	ACKNOWLEDGED
13	LOW	Token could turn into a partial honeypot if the liquify threshold is ever set to zero	✓ RESOLVED
14	INFO	Lack of events for `setSelling`, `setBuying`, `setMarketingWallet`, `setBTCBRewardsFee`, `setLiquidityFee`, `setMarketingFee`, `blacklistAddress`, `withdrawShiva` and `withdrawBNB`	✓ RESOLVED
15	INFO	Usage of `.transfer()` instead of `.call()` to send BNB	ACKNOWLEDGED
16	INFO	Many functions can be made `external`	PARTIAL
17	INFO	withdrawBNB is wrongly defined	ACKNOWLEDGED

1.3.2 SHIVADividendTracker

ID	Severity	Summary	Status
18	HIGH	The minimum amount of tokens an account has to hold to be eligible for dividends can be set excessively high to half of the eligible supply	✓ RESOLVED
19	INFO	Adjusting the minimum does not affect users until they make a transaction	ACKNOWLEDGED
20	INFO	Wrongful usage of require instead of revert	✓ RESOLVED
21	INFO	Token symbol exceeds 11 characters which makes adding it to MetaMask more cumbersome	✓ RESOLVED
22	INFO	process and getAccountAtIndex can be made external	✓ RESOLVED
23	INFO	processesUntilEndOfArray is a misnomer	✓ RESOLVED
24	INFO	Lack of event for process function	✓ RESOLVED
25	INFO	Owner can give themself a dividend receiving position before ownership is transferred	✓ RESOLVED

1.3.3 DividendPayingToken

ID	Severity	Summary	Status
26	LOW	The success check in `distributeBTCBDividends` is insufficient since BTCB transfers will revert instead of returning false which could cause transfers to block if there is ever insufficient BTCB in the contract	ACKNOWLEDGED
27	LOW	`distributeBTCBDividends` does not verify that enough BTCB has been deposited into the contract, which could block transfers and withdrawals	ACKNOWLEDGED
28	INFO	Owner can give themself a dividend receiving position before ownership is transferred	✓ RESOLVED
29	INFO	Wrongful usage of require instead of revert	✓ RESOLVED
30	INFO	BTCB can be made constant	✓ RESOLVED

1.3.4 IterableMapping

No issues found.

2 Findings

2.1 SHIVA

The SHIVA token is a token which uses its transfer tax to generate liquidity, generate BTCB dividends, swaps to BNB that can be used for automated buybacks and as a fee to the marketing wallet. Finally a cool down period can be set to prevent swaps for a certain period for a wallet that has made a swap.

The owner, burn wallet, dividend tracker, token itself and router are excluded from receiving dividends by default. Furthermore, the owner, marketing wallet and the token itself are excluded from paying fees by default. The marketing wallet, burn wallet and the token address itself are excluded from the anti-whale. Finally the owner, marketing address, burn address and the token itself are excluded from the swap cooldown period. These can be changed freely by the governance.

Initially there's a 10% BTCB rewards fee, a 5% liquidity fee, a 5% marketing fee and a 5% buyback fee on sell. These fees can be adjusted freely. Generated liquidity is burned.

A total of 51 billion tokens are minted to the owner during deployment, which accounts for the total supply.

Swaps are not allowed until the `swapStartBlock` has been reached. Once it has been reached, this block can no longer be moved. It is initially set to 91651040.

2.1.1 Privileged Roles

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

- `withdrawShiva`
- `withdrawBNB`
- `excludeFromFees`
- `setExcludedFromAntiWhale`
- `excludeMultipleAccountsFromFees`
- `updateMaxTransferAmountRate`
- `updateMaxSaleAmountRate`
- `updateLimitSwap`
- `updateSwapAndLiquifyDividendEnabled`
- `updateLimitSwapTime`
- `updateSwapStartTime`
- `setMarketingWallet`
- `setBTCBRewardsFee`
- `setLiquidityFee`
- `setMarketingFee`
- `setAutomatedMarketMakerPair`
- `updateGasForProcessing`
- `updateClaimWait`
- `updateMinimumTokenBalanceForDividends`

2.1.2 Issues & Recommendations

Issue #01	Typographical errors cause contract to fail compilation
Severity	● HIGH SEVERITY

Description	Line 1231
	`uint16 public maxTransferAmountRate = 50`
	Line 1231 is not terminated by a semicolon which causes compilation to fail. This prevents the contract from being deployed or even tested until this is resolved.
	
	Line 1310
	`if ( automatedMarketMakerPairs[from] ) {`
	
	Line 1314
	`if ( automatedMarketMakerPairs[to] ) {`
	These should be using sender and recipient.
	
	Line 1289
	`event updateMaxSellAmount(address indexed operator, uint256 previousAmount, uint256 newAmount);`
	
	Line 1431
	`function updateMaxSellAmount(uint16 _maxSellAmount) public onlyOwner {`
	Consider renaming the event to event `MaxSellAmountUpdated(...);`

Recommendation	Consider fixing the typographical errors.
Resolution	✓ RESOLVED The typographical errors have been resolved and the contract can now be compiled.

Issue #02	DoS: Tokens can be sent to a user periodically to trigger their swap limit and prevent them from making any transactions
Severity	● HIGH SEVERITY
Description	The token contains a cooldown functionality that prevents further purchases and sales after one has occurred. However, a malicious party can let the uniswap pair send a small amount of tokens to the user periodically to block their wallet from creating any transactions by themselves.
Recommendation	Consider removing the swap lock functionality completely.
Resolution	● ACKNOWLEDGED

Issue #03	Gov Privilege: `updateUniswapV2Router` could be used to revert transactions, siphon fees or turn the token into a honeypot
Severity	● HIGH SEVERITY
Description	The owner can update the router that generates liquidity to an address or contract of choice. This contract could be a malicious contract that simply keeps the tokens sent to it and thus siphons all deposit fees. Furthermore this contract could be used to revert sell transactions turning the token into a honeypot.
Recommendation	Consider removing this function. If this is not possible, consider using an operator account which is behind a significantly longer timelock so investors can reasonably see this change coming and inspect the new router.
Resolution	✓ RESOLVED The `update` function has been removed.

Issue #04	Gov privilege: Maximum buy and sell amounts can be set freely to potentially turn the token into a honeypot
Severity	● HIGH SEVERITY
Description	The token defines both a maximum sell and maximum buy amount, individual purchases and sales from the main pair cannot exceed this amount. The token could therefore be turned into a honeypot by setting the maximum sell amount to zero while maintaining a large maximum buy amount.
Recommendation	Consider removing this functionality or adding reasonable minima to these variables.
Resolution	✓ RESOLVED This limit must now at least be set to 0.1% of the total supply. It should be noted that this could still be low in case the token has a low market capitalization.

Issue #05	Gov privilege: Governance can blacklist wallets preventing them from making any further transactions like selling their tokens
Severity	● HIGH SEVERITY
Description	The contract governance can blacklist wallets which prevents these wallets from creating any sort of transaction. This could be abused by the governance by blocking wallets as soon as they make a large purchase, turning the token effectively into a honeypot.
Recommendation	Consider removing the blacklisting functionality.
Resolution	✓ RESOLVED The `blacklistAddress` function has been removed.

Issue #06	Maximum buy and sell amount can only be set to infinitesimally small amounts
Severity	● HIGH SEVERITY
Description	The code contains governance functionality to update the maximum sell and buy amounts an account can do within a single transaction, however, the type of these amounts is set to uint16, which has a maximum of 65535, an extremely small portion of the total supply. ❗ The selling and buying parameters can furthermore be set to false to turn the token into a honeypot.
Recommendation	Consider removing the maximum buy and sell amount functionality completely since it has so many side-effects. If this is not possible, consider making the parameter uint256 and adding a very reasonable minimum. Furthermore consider removing the selling and buying parameters which can also be abused to turn the token into a honeypot.
Resolution	✓ RESOLVED These are now expressed as rates.

Issue #07	Gov Privilege: Fees are freely adjustable up to over 100%
Severity	● HIGH SEVERITY
Description	The owner of the contract can set the individual fees to any variable at all. This might deter investors as they could be scared that these fees might one day be set to 100% to force transfers into the contract owner.

Recommendation	Consider adding an explicit cap to the total fee on every fee adjustment function. The example below requires the total fee to be less than 20%.
	
	`totalFees =`
	`BTCBRewardsFee.add(liquidityFee).add(marketingFee);`
	`require(totalFees <= 20, “too high”);`
	
	This issue will also be marked as resolved once disableFeeChanging is called.

Resolution	✓ RESOLVED The recommendation has been added, limiting the total fee to 20%.

Issue #08	Gov Privilege: Owner can update the dividend tracker to siphon all dividends and potentially block sell transactions
Severity	● MEDIUM SEVERITY
Description	Currently the owner of the Shiva token can freely upgrade to a new underlying dividend tracker. If this is done to a malicious tracker it could block sell transactions (through `swapAndSendDividends`) and siphon the BTCB dividends to the owner instead of distributing them. This privilege could harm investor confidence.
Recommendation	Consider removing the `updateDividendTracker` function if there is no use of upgradeability. Otherwise consider a significant timelock.
Resolution	✓ RESOLVED The `updateDividendTracker` function has been removed.

Issue #09	Gov privilege: Maximum anti-whale transfer amount can be set as low as 0.0001% of the total supply, practically disabling all transfers
Severity	● MEDIUM SEVERITY
Description	The token includes functionality that limits the maximum transfer size of any token transfer. However, this limit can be set as low as 0.0001% of the total supply which practically disables all transfer functionality.
Recommendation	Consider removing this functionality or adding a more reasonable minimum like 1%.
Resolution	✓ RESOLVED A minimum of 0.1% has been instated, this might still be considered small.

Issue #10	Gov privilege: If the marketing wallet is a contract, it could reject the BNB transfers turning the token into a honeypot
Severity	● MEDIUM SEVERITY
Description	The marketing wallet can be set to a contract which can reject BNB transfers in its fallback function, this would revert all BNB transfers to the marketing wallet which can happen on all transactions except purchases, therefore potentially turning the token into a honeypot.
Recommendation	Consider using WETH and not allowing for the marketing wallet to be set not zero.
Resolution	● ACKNOWLEDGED The client has not allowed zero-transfers to the marketing wallet but this wallet can still be set to the zero address. Furthermore, `.transfer` is still used which makes the function revertable.

Issue #11	`limitSwap` functionality is broken
Severity	● MEDIUM SEVERITY

Location	Line 1622-1628
	`uint256 lastSwap = _userInfo[userAddress];`
	`uint256 checkLastSwap = block.number.sub(lastSwap);`
	`if(_excludedLimitSwap[userAddress] == false){`
	`    require(checkLastSwap >= timeLimitSwap, "SHIVA:: Trade Too fast");`
	`} else {`
	`    _userInfo[userAddress] = block.number;`
	`}`

Description	The governance can enable `limitSwap`, which creates a cool down period between individual swaps during which no further swaps can be made by the user. However, when users are not excluded from the `limitSwap`, their `_userInfo` is actually never updated to `block.number`. This means that the limit swap does not apply.
Recommendation	Consider simply removing the limit swap functionality altogether as it causes many issues even if it would work.
Resolution	✓ RESOLVED The last swap is now always updated.

Issue #12	Gov privilege: Governance can exclude wallets from receiving dividends
Severity	● LOW SEVERITY
Description	The governance can exclude specific wallets from receiving dividends which could be abused by excluding people they dislike.
Recommendation	Consider putting the ownership of the contract behind a timelock so people can inspect these transactions and act accordingly.
Resolution	● ACKNOWLEDGED This issue will be marked as resolved once we confirm that this has been locked behind a reasonable timelock.

Issue #13	Token could turn into a partial honeypot if the liquify threshold is ever set to zero
Severity	● LOW SEVERITY
Description	The token will attempt to swap liquidity once the `swapTokensAtAmount` threshold is reached in fees collected. However, if this variable is set to zero, this threshold will be reached even though there are no tokens within the router. Therefore, the contract will currently attempt a swap and liquidity addition and Uniswap-like AMMs will revert due to the lack of input tokens.
Recommendation	Consider adding a minimum to the `swapTokensAtAmount` threshold and furthermore wrapping the Uniswap operations within try-catch statements.
Resolution	✓ RESOLVED This function is not configurable and set to non-zero from the start.

Issue #14	Lack of events for `setSelling`, `setBuying`, `setMarketingWallet`, `setBTCBRewardsFee`, `setLiquidityFee`, `setMarketingFee`, `blacklistAddress`, `withdrawShiva` and `withdrawBNB`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events to the above functions.
Resolution	✓ RESOLVED

Issue #15	Usage of `.transfer()` instead of `.call()` to send BNB
Severity	INFORMATIONAL
Description	The contract uses `.transfer` instead of `.call` to transfer BNB, it has been documented that `.transfer` could potentially break on future hard forks due to it having a very restricted gas limit.
Recommendation	Consider using `.call` instead.
Resolution	● ACKNOWLEDGED

Issue #16	Many functions can be made `external`
Severity	● INFORMATIONAL
Description	A large portion of the functions can be made `external`, which signifies that they are not used within the contract themselves.
Recommendation	Consider marking all functions that are not used within the contract but only externally as external.
Resolution	● PARTIALLY RESOLVED Some of these functions have been made external.

Issue #17	`withdrawBNB` is wrongly defined
Severity	● INFORMATIONAL

Location	Lines 1822-1829
	`function withdrawBNB(address toAddress, uint256 amount) external onlyOwner {`
	`    uint256 bnbblance = address(this).balance;`
	`    if(bnbblance <= amount) {`
	`        amount = bnbblance;`
	`    }`
	`    payable(toAddress).transfer(bnbblance);`
	`    emit BNBWithdrawn(msg.sender, toAddress, amount);`
	`}`

Description	The `withdrawBNB` function withdraws `bnbbalance` instead of `amount` and the `amount` parameter is therefore redundant.
Recommendation	Consider fixing the function similar to the withdrawShiva function.
Resolution	● ACKNOWLEDGED

2.2 SHIVADividendTracker

The SHIVADividendTracker contract extends the DividendPayingToken contract to automate the distribution of BTCB to token holders.

A minimum of 100,000 tokens is necessary for users to be eligible for a dividend. This minimum can be changed by the admin and can be between 100% and 50% of the total supply. Users are only eligible for automatic distribution every 6 hours (can be adjusted up to 24 hours), however, they can still claim their BTCB distributions manually using the claim function.

2.2.1 Privileged Roles

The following functions can be called by the owner of the contract, which will eventually be the SHIVA contract:

- `setBalance`
- `processAccount`

2.2.2 Issues & Recommendations

Issue #18	**The minimum amount of tokens an account has to hold to be eligible for dividends can be set excessively high to half of the eligible supply**
Severity	● HIGH SEVERITY
Location	Line 1902 `require(newMinimumTokenBalanceForDividends >=100* (10**18) && newMinimumTokenBalanceForDividends <=totalSupply().div(2), "SHIVA_Dividend_Tracker: MinimumTokenBalanceForDividends must be updated to between 100 and half of totalsupply");`
Description	The owner has the ability to adjust the minimum amount of tokens that an account has to hold to be eligible for dividends up to half of the eligible supply. Under this parameterization normal investors that do not hold half of the supply will be unable to ever get dividends. ❗ It should be noted that the totalsupply of the dividend tracker is used and not of the actual token, this means only tokens that are actually eligible for dividends are accounted in this totalsupply.
Recommendation	Consider using a more reasonable maximum, eg. 10 million tokens.
Resolution	✓ RESOLVED A fixed limit of 10 million tokens has been instated.

Issue #19	Adjusting the minimum does not affect users until they make a transaction
Severity	● INFORMATIONAL

Locations	Line 1901
	`function updateMinimumTokenBalanceForDividends(uint256 newMinimumTokenBalanceForDividends) external onlyOwner {`
	
	Line 1993-2000
	`if(newBalance >= minimumTokenBalanceForDividends) {`
	`    _setBalance(account, newBalance);`
	`    tokenHoldersMap.set(account, newBalance);`
	`}`
	`else {`
	`    _setBalance(account, 0);`
	`   tokenHoldersMap.remove(account);`
	`}`

Description	The owner has the ability to adjust the minimum amount of tokens that an account has to hold to be eligible for dividends. However, any existing account their balance will not be nullified or set to the correct balance if it leaves or enters eligibility. In case the minimum is raised, accounts will remain eligible until they make a transaction, in case it is lowered, accounts remain excluded until they make a transaction.
Recommendation	Consider using a non-changeable minimum.
Resolution	● ACKNOWLEDGED

Issue #20	Wrongful usage of `require` instead of `revert`
Severity	● INFORMATIONAL

Location	`Line 1877`
	`require(false, "SHIVA_Dividend_Tracker: No transfers allowed");`
	
	`Line 1881`
	`require(false, "SHIVA_Dividend_Tracker: withdrawDividend disabled. Use the 'claim' function on the main SHIVA contract.");`

Description	To make sure that people cannot manually transfer the token, a `require(false)` statement is added to the transfer override function. This will always revert said function. However, for this behavior, Solidity recommends using the `revert();` keyword instead, since `require` is meant to actually require things to be true.
Recommendation	Consider using `revert(“reason”);` instead.
Resolution	✓ RESOLVED

Issue #21	Token symbol exceeds 11 characters which makes adding it to MetaMask more cumbersome
Severity	● INFORMATIONAL
Description	Although the ERC-20 metadata standard does not specify a maximum length for a token symbol, MetaMask does not allow the length to exceed 11 characters. Adding any token to MetaMask with a symbol that is over 11 characters will require the user to manually adjust the symbol, which could be considered bad UX.
Recommendation	Consider whether it is possible to remove letters from the symbol string to make it compliant with MetaMask without user intervention.
Resolution	✓ RESOLVED

Issue #22	**`process` and `getAccountAtIndex` can be made `external`**
Severity	● INFORMATIONAL
Description	The `process` and `getAccountAtIndex` functions can be changed from `public` to `external`. 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 making these functions external.
Resolution	✓ RESOLVED

Issue #23	**`processesUntilEndOfArray` is a misnomer**
Severity	● INFORMATIONAL

Location	Lines 605-607
	`uint256 processesUntilEndOfArray = tokenHoldersMap.keys.length > lastProcessedIndex ?`
	`    tokenHoldersMap.keys.length.sub(lastProcessedIndex) :`
	`    0;`

Description	The variable `processesUntilEndOfArray` is wrongly called like this since it actually keeps track of the processes until the beginning of the array. Take for example an array of 3 holders and we are already at the last index, index 2. The variable will in this case indicate that there is still 1 process to go.
Recommendation	Since the business logic actually matches the purpose of this variable, the variable should simply be renamed to `processesUntilbeginOfArray`.
Resolution	✓ RESOLVED

Issue #24	Lack of event for `process` function
Severity	● INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. In this case, we believe it might be valuable to emit an event for a whole batch process call that indicates from and to which index the processing occurred.
Recommendation	Add an event for the above function.
Resolution	✓ RESOLVED

Issue #25	Owner can give themself a dividend receiving position before ownership is transferred
Severity	● INFORMATIONAL
Description	The owner of the ShivaDividendTracker can manually give shares to accounts to receive a share of the dividends. Usually this is the SHIVA contract, however, SHIVA contains a function `updateDividendTracker` which allows for the moving of the token to a new token, and this new token could then have premined balances. Pre-mined balances would result in the owner taking a part of the BTCB dividends as long as they have the balance.
Recommendation	Consider removing the `updateDividendTracker` function or putting it behind an extremely long timelock for this issue to be marked as resolved.
Resolution	✓ RESOLVED The `updateDividendTracker` function has been removed.

2.3 DividendPayingToken

The DividendPayingToken is a token contract that allows for the distribution of BTCB dividends to the token holders. Dividends need to be sent to it by the contract owner which should be the SHIVA contract.

2.3.1 Privileged Roles

- `distributeBTCBDividends`

2.3.2 Issues & Recommendations

Issue #26	**The success check in `distributeBTCBDividends` is insufficient since BTCB transfers will revert instead of returning false which could cause transfers to block if there is ever insufficient BTCB in the contract**
Severity	● LOW SEVERITY

Location	`Line 1824`
	`bool success = IERC20(BTCB).transfer(address(dividendTracker), dividends);`

Description	When `distributeBTCBDividends` is called by the owner of the contract to distribute the dividend, this does not explicitly ensure that enough BTCB was actually deposited into the contract. This issue has been lowered to low severity since the SHIVA contract should own the tracker, forcing this logic to always be correct.
Recommendation	Consider adding try-catch logic to the transfer call to also handle the failure case when the transfer does not succeed and reverts. This way token transfers are not blocked due to there being insufficient BTCB in the contract.
Resolution	● ACKNOWLEDGED Although the try-catch logic is implemented, a revert statement is added into the catch making it redundant.

Issue #27	**`distributeBTCBDividends` does not verify that enough BTCB has been deposited into the contract, which could block transfers and withdrawals**
Severity	● LOW SEVERITY

Location	`Line 1101`
	`function distributeBTCBDividends(uint256 amount) public onlyOwner{`

Description	When `distributeBTCBDividends` is called by the owner of the contract to distribute the dividend, this does not explicitly ensure that enough BTCB was actually deposited into the contract.
Recommendation	Consider either verifying that enough BTCB was added to the contract by for example pulling it in or consider the recommendations from the previous issue to not make transfers fail when the BTCB transfer is unsuccessful.
Resolution	● ACKNOWLEDGED

Issue #28	**Owner can give themself a dividend receiving position before ownership is transferred**
Severity	● INFORMATIONAL
Description	The owner of the DividendPayingToken can manually give shares to accounts to receive a share of the dividends. Usually this is the SHIVA contract, however, SHIVA contains a function `updateDividendTracker` which allows the moving of the token to a new dividend tracking token, and this new token could then have pre-mined balances. Premined balances would result in the owner taking a part of the SHIVA dividends as long as they have the balance.
Recommendation	Consider removing the `updateDividendTracker` function or putting it behind an extremely long timelock for this issue to be marked as resolved.
Resolution	✓ RESOLVED Within SHIVA, the `updateDividendTracker` has been removed.

Issue #29	Wrongful usage of `require` instead of `revert`
Severity	INFORMATIONAL
Location	Line 1175 `require(false);`
Description	To make sure that people cannot manually transfer the token, a `require(false)` statement is added to the transfer override function. This will always revert said function. However, for this behavior, Solidity recommends using the `revert();` keyword instead, since `require` is meant to actually require things to be `true`.
Recommendation	Consider using `revert();` instead.
Resolution	✓ RESOLVED

Issue #30	BTCB can be made `constant`
Severity	INFORMATIONAL
Description	Variables that are not changed throughout the contract can be marked as constant. This makes it easier for third-party reviewers to understand the contract and might reduce gas usage.
Recommendation	Consider marking the variable as `constant`.
Resolution	✓ RESOLVED

2.4 IterableMapping

The IterableMapping library is a dependency which allows for the creation of Solidity key-value mappings which are iterable as well.

2.4.1 Issues & Recommendations

No issues found.

PALADIN

BLOCKCHAIN SECURITY