PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For GATORVaults

11 July 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 GATORVaults. 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 Important Note

Paladin has audited GatorVaultV2 and StrategyGatorMaxi with the assumption that GatorVaultV2 uses the StrategyGatorMaxi and deposits in the RewardPool contract that has also been audited in this report. This strategy allows users to compound VGator tokens automatically over time, selling off the reward BNB earned from the RewardPool contract to stake more Gator.

1.2 Summary

Project Name	GATORVaults
URL	https://gatorvaults.com/
Platform	Binance Smart Chain
Language	Solidity

1.3 Contracts Assessed

The contracts below have been provided to Paladin after resolving the issues raised in the preliminary report and the client has confirmed that these are the actual contracts to be used in their project.

Paladin has gone through the contracts and verified that the code in the contracts match the ones they have sent to us after applying code fixes.

Name	Contract	Live Code Match
GatorVaultV2	0x7ED57753bcB18898BA69B9F422d28A7e79fe1222	MATCH
StrategyVGatorMaxi	0xF262ECf367Ae2DA78518160f64c37a9ad5e7830d	MATCH
RewardPool	0x5385eD13331A2C74Ec8E1E5eE7A644A6DB2ECDdE	MATCH
VGator	0xF621D455f803c65F6C19a7e9cd6bEfA93bb7Dbd0	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	3	3	-	-
Low	2	2	-	-
Informational	16	14	-	2
Total	**24**	**22**	**0**	**2**

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 GatorVaultV2

ID	Severity	Summary	Status
01	HIGH	Vault code is instantly upgradeable	RESOLVED
02	MEDIUM	Deposit fee issue still present even though it looks accounted for on the surface	RESOLVED
03	LOW	Consistency: `uint256` is not used everywhere	RESOLVED
04	INFORMATIONAL	Gas optimization: `withdraw` function could be simplified	ACKNOWLEDGED
05	INFORMATIONAL	Lack of events for `deposit` and `withdraw`	RESOLVED
06	INFORMATIONAL	Ownership can be renounced after initialization	RESOLVED

1.3.2 StrategyGatorMaxi

ID	Severity	Summary	Status
07	INFORMATIONAL	`rewardsFee` seems redundant	RESOLVED
08	INFORMATIONAL	Unused interfaces and contracts `IWBNB` and `ERC20`	RESOLVED
09	INFORMATIONAL	`swapExactTokensForTokens` will no longer work if this contract is ever reused for a transfer tax reward token	RESOLVED
10	INFORMATIONAL	Unnecessary timestamp modification in the `swapRewards` call	RESOLVED
11	INFORMATIONAL	The VGatorToken address is hardcoded in the contract	RESOLVED
12	INFORMATIONAL	Lack of events for `harvest`, `retireStrat` and `panic`	RESOLVED

1.3.3 RewardPool

ID	Severity	Summary	Status
13	HIGH	The `LPTokenWrapper` does not account for transfer tax which could lead to the pool draining over time	RESOLVED
14	MEDIUM	`exit` and `getReward` will revert if there is not enough wbnb in the RewardPool	RESOLVED
15	MEDIUM	`notifyRewardAmount` with an excessively high reward amount could block all `stake`, `withdraw`, `exit` and `getReward` calls forever	RESOLVED
16	INFORMATIONAL	The VGatorToken address is hardcoded in the contract	RESOLVED
17	INFORMATIONAL	Lack of event for `setRewardDistribution`	RESOLVED

1.3.4 VGatorToken

ID	Severity	Summary	Status
18	HIGH	`updateGatorSwapRouter` could be used by the project team to siphon all swap fees or block deposits	RESOLVED
19	LOW	LP tokens are sent to the operator which could remove and sell them	RESOLVED
20	INFORMATIONAL	`swapAndLiquify` could fail if there are no tokens to add to liquidity	RESOLVED
21	INFORMATIONAL	`delegateBySig` can be frontrun and cause denial of service	ACKNOWLEDGED
22	INFORMATIONAL	Deployer account is excluded from tax	RESOLVED
23	INFORMATIONAL	Lack of events for `includeInTax` and `excludeFromTax`	RESOLVED
24	INFORMATIONAL	`includeInTax`, `excludeFromTax`, `transferOperator`, `updateGatorSwapRouter`, `updateSwapAndLiquifyEnabled`, `updateMinAmountToLiquify`, `updateBurnRate` and `updateTransferTaxRate` can be made `external`	RESOLVED

2 Findings

2.1 GatorVaultV2

The GatorVaultV2 is a simple vault based on the Beefy vault pattern.

2.1.2 Privileged Roles

The following `onlyOwner` functions can be called by the owner:

- `upgradeStrat`
- `proposeStrat`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	Vault code is instantly upgradeable
Severity	HIGH SEVERITY
Description	While the vault code appears to be upgradeable only once due to it updating the `stratCandidate.proposedTime` to a year equivalent to 2128, in reality this lock does not apply as it is overridden within `proposeStrat`. This currently allows the owner to upgrade the vault whenever they want, allowing them to steal the underlying tokens. This final step is done through upgrading to a malicious vault, getting a high balance for free in this vault, and downgrading to the innocent vault containing user funds again. Because the malicious account now has an inflated balance, they can withdraw all user funds.
Recommendation(s)	Consider removing the strategy parameter from the constructor and using the following function to initialize the strategy exactly once:

```
bool public initialized = false;

// InitializeStrat allows the owner to upgrade the strategy once for the initial
deployment of the vault.

function initializeStrat(address implementation) public onlyOwner {
    require(!initialized, "The strategy has already been set");
    require(totalSupply() == 0 && strategy == address(0));

    initialized = true;
    strategy = implementation;
    emit UpgradeStrat(implementation);
}
```

The event is kept to remain consistent with potential programs that listen for Beefy events.

Resolution	RESOLVED The client has removed the upgradeability code for a one-time initialisable vault. Safety checks have been added to ensure that nobody has deposited until the vault is initialized.

Issue #02	Deposit fee issue still present even though it looks accounted for on the surface
Severity	MEDIUM SEVERITY
Description	In many contracts, the contract improperly handles the transfer of tokens with a deposit fee. The contract accounts for the tokens sent instead of the tokens received. The GatorVaultV2 seems to solve this on the surface using the recommended before-after pattern. However, since later on the funds have to still be transferred to the vault, new deposits would still reduce the balance of all previous stakers.
Recommendation(s)	Consider fundamentally rethinking the way the vault operates or simply not using transfer tax tokens.
Resolution	RESOLVED The client has confirmed that they have excluded all relevant contracts from the Gator transfer tax, which is the only token that will be used in these vaults.

Issue #03	Consistency: `uint256` is not used everywhere
Severity	LOW SEVERITY
Description	Throughout the contract, sometimes `uint` is used instead of `uint256`. This can be misleading for reviewers as it is inconsistent.
Recommendation(s)	Consider replacing all `uint` occurrences with `uint256`.
Resolution	RESOLVED All occurences of `uint` have been replaced with `uint256`.

Issue #04	Gas optimization: Withdraw function could be simplified
Severity	INFORMATIONAL
Description	In the withdraw function, a before-after pattern is done to calculate how much tokens are withdrawn from the strategy. Then `r`, the amount to be sent to the user, is set to `b.add(_diff)`. This could be simplified to save gas.

Recommendation(s)	Consider simplifying the final assignment of `r`:
	`r = b.add(_diff);`
	`r = _after;`

Resolution	ACKNOWLEDGED Client prefers not to change the code to avoid unintended side-effects.

Issue #05	Lack of events for `deposit` and `withdraw`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation(s)	Add events for `deposit` and `withdraw`.
Comments	RESOLVED Events added for `deposit` and `withdraw`.

Issue #06	Ownership can be renounced after initialization
Severity	INFORMATIONAL
Description	Since there are no more owner privileges after the initialisation, this can be communicated to investors by setting the owner address to the zero address. Many investors will immediately understand that there are no more ownership privileges on this vault when they see that the ownership is renounced.
Recommendation(s)	Renounce ownership after initializing the strategy.
Comments	RESOLVED The client has said they will renounce ownership after initialization to communicate this to investors.

2.2 StrategyGatorMaxi

The StrategyGatorMaxi is the underlying strategy for GatorVaultV2 and is based on Beefy. It stakes in the RewardPool to sell the earned BNB to continuously compound more VGator tokens.

There is a fixed withdrawal fee of 0.05% which cannot be increased.

The strategy also has two fixed performance fees which are a percentage of the harvest rewards:

- A call fee of 0.5% which is sent to the owner when they harvest the strategy
- A rewards fee of 0.5% which is sent back to the rewards contract

The strategy can be paused from depositing and harvesting, but withdrawals are always possible as long as the underlying contract continues functioning.

2.2.1 Privileged Roles

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

- `unpause`
- `pause`
- `panic`
- `retireStrat`
- `harvest`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #07	`rewardsFee` seems redundant
Severity	INFORMATIONAL
Description	When the strategy rewards are harvested from the underlying RewardPool, 0.5% is sent back to the underlying RewardPool. This looks like an unnecessary operation to us as the RewardPool manager might as well have sent 99.5% of the rewards to the RewardPool and an extra 0.5% in the next round. Furthermore, this reward is only activated when the owner actually informs the RewardPool about it, which might be difficult as they have to figure out how much was actually transferred to the underlying rewardPool.
Recommendation(s)	Consider removing the `rewardsFee`.
Resolution	RESOLVED The `rewardsFee` has been removed completely.

Issue #08	Unused interfaces and contracts `IWBNB` and `ERC20`
Severity	INFORMATIONAL
Description	The contract contains unused interfaces which make it unnecessary longer.
Recommendation(s)	Consider removing `IWBNB` and `ERC20`.
Resolution	RESOLVED The unused libraries have been removed.

Issue #09	`swapExactTokensForTokens` will no longer work if this contract is ever reused for a transfer tax reward token
Severity	INFORMATIONAL
Description	In `swapRewards` function, the Uniswap V1 `swapExactTokensForTokens` function is used, this function will fail if the wbnb token is ever replaced with a reward token that has a transfer tax.
Recommendation(s)	Consider this behavior if this contract is ever reused with an underlying reward with a transfer tax.
Resolution	RESOLVED The team has assured us that they will not use this contract for any token but WBNB.

Issue #10	Unnecessary timestamp modification in the `swapRewards` call
Severity	INFORMATIONAL
Description	Uniswap transactions will still pass if the deadline is the same as the block timestamp, the `.add(600)` is thus unnecessary boilerplate code.
Recommendation(s)	Consider removing the `.add(600)` from the `swapRewards` function to simplify the code further.
Comments	RESOLVED The boilerplate code has been removed to simplify the contract further.

Issue #11	The VGator address is hardcoded in the contract
Severity	INFORMATIONAL
Description	The VGoofyToken address is hardcoded in the contract. If the project is redeployed without changing this, the contract would not function.
Recommendation(s)	Remember to update this address during the production deployment, or add it as a constructor parameter.
Comments	RESOLVED The token and rewards contracts are now set in the constructor during deployment.

Issue #12	Lack of events for `harvest`, `retireStrat` and `panic`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation(s)	Add events for `harvest`, `retireStrat` and `panic`.
Comments	RESOLVED Events added to `harvest` and `panic`, and `retireStrat` has been removed completely to further remove governance privileges and is no longer necessary. This function thus no longer needs an event.

2.3 RewardPool

The RewardPool is a staking contract that allows users to stake VGator and earn BNB. It is simplified compared to other contracts, most notably in the fact that the owner cannot take out the BNB once transferred in.

2.3.1 Privileged Roles

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

- `transferOwnership`
- `renounceOwnership`
- `setRewardDistribution`

2.3.2 Issues & Recommendations

Issue #13	The `LPTokenWrapper` does not account for transfer tax which could lead to the pool draining over time
Severity	HIGH SEVERITY
Description	When a token is staked, the transfer tax of the VGator token is not accounted for. This means that the rewardPool will receive less VGator tokens then are sent to it. Over time, this will result in the pool being drained of VGator.
Recommendation(s)	Consider using a before-after pattern:

```
uint256 balanceBefore = pool.lpToken.balanceOf(address(this));
pool.lpToken.transferFrom(msg.sender, address(this), _amount);
amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);
```

Excluding the RewardPool from the fee also works.

Resolution	RESOLVED The recommended pattern has been implemented and the team has reassured us that the token will also never have a transfer tax on this system as all relevant contracts are excluded from the tax.

Issue #14	exit and getReward will revert if there is not enough wbnb in the RewardPool
Severity	MEDIUM SEVERITY
Description	The exit and getReward functions will revert when there is not enough wbnb in the pool to be sent out as a reward to stakers. This happens specifically when not enough wbnb is transferred in together with notifyRewardAmount.
Recommendation(s)	Consider having a `safeWBNBTransfer` function similar like the one in the MasterChef:

```
function safeWBNBTransfer(address _to, uint256 _amount) internal {
  uint256 balance = wbnb.balanceOf(address(this));

  if (_amount > balance) {
      wbnb.safeTransfer(_to, balance);
  } else {
      wbnb.safeTransfer(_to, _amount);
  }
}
```

A second safety measure could be to use `safeTranserFrom` in the `notifyRewardAmount` function.

Resolution	RESOLVED The recommended `safeWBNBTransfer` function has been added.

Issue #15	`notifyRewardAmount` with an excessively high reward amount could block all `stake`, `withdraw`, `exit` and `getReward` calls forever
Severity	MEDIUM SEVERITY
Description	Due to SafeMath reverting on overflow, an excessively high `rewardRate` will revert all `rewardPerToken()` calls. This means that deposited funds will be stuck forever. Although this seems an unlikely case, it could occur if the developer turns malicious (even though there is no economic incentive) but more importantly by accident when a wrong value is sent by accident due to mismanagement of the timelock. We are aware of multiple occurrences of this latter case.
Recommendation(s)	Consider adding a safeguard to the maximum value of reward in `notifyRewardAmount`.
Resolution	RESOLVED `notifyRewardAmount` can be called in increments of up to approximately 50 WBNB.

Issue #16	The VGatorToken address is hardcoded in the contract
Severity	INFORMATIONAL
Description	The VGoofyToken address is hardcoded in the contract. If the project is redeployed without changing this the contract would not function.
Recommendation(s)	Remember to update this address during the production deployment, or add it as a constructor parameter.
Resolution	RESOLVED The token is now set in the constructor during deployment.

Issue #17	Lack of event for `setRewardDistribution`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation(s)	Add event for `setRewardDistribution`.
Resolution	RESOLVED

2.4 VGator

VGator is a token contract forked from PantherSwap. Initially, it has a 5% transfer tax, and can be set up to a maximum of 10%. Of this transfer tax, 20% is burned and 80% is used to generate liquidity, and these percentages are adjustable by the operator of the contract. There is no anti-whale mechanism which is not an issue as advanced players can easily circumvent it.

2.4.1 Token Overview

Address	0xF621D455f803c65F6C19a7e9cd6bEfA93bb7Dbd0
Token Supply	1 billion (1,000,000,000)
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	5% (up to maximum 10%)

2.4.2 Privileged Roles

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

- `renounceOwnership`
- `transferOwnership`
- `mint`
- `excludeFromTax`
- `includeInTax`

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

- `updateTransferTaxRate`
- `updateBurnRate`
- `updateMinAmountToLiquify`
- `updateSwapAndLiquifyEnabled`
- `updateGoofySwapRouter`
- `transferOperator`

2.4.3 Issues & Recommendations

Issue #18	`updateGatorSwapRouter` could be used by the project team to siphon all swap fees or block deposits
Severity	HIGH SEVERITY
Description	The owner can update the Uniswap router address that is responsible for converting the transfer fee into BNB for liquidity. When this router is changed to a malicious contract, this could be used to block sell transactions (thus turning it into a honeypot). It could also be used to simply keep the transaction fees and send them to the owner.
Recommendation(s)	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 client has said they will transfer token operatorship to an 8 day timelock. Although we recommend investors to still verify that this has been done, an 8 day timelock is exceptional in the industry and should give investors plenty of time to react to this potential change while retaining the freedom for the project to one day move to their own exchange.

Issue #19	LP tokens are sent to the operator which could remove and sell them
Severity	LOW SEVERITY
Description	The LP tokens generated through the liquidity system are sent to the operator (project owner). This gives the operator the ability to remove and cash out the liquidity which can be bad for investor confidence.
Recommendation(s)	Consider transferring these to a vesting contract or simply sending them to the burn address. A third option which may align better with tokenomics is to keep them in the contract and add a manual `buybackAndBurn` function which removes a small part of the liquidity for an automated buy back and burn. It should be noted that if too much slippage is created through this buyback, front-running bots might take a piece of it.
Resolution	RESOLVED The client has updated the function to now send these LP tokens to the burn address, these LPs are locked forever.

Issue #20	`swapAndLiquify` could fail if there are no tokens to add to liquidity
Severity	LOW SEVERITY
Description	Within the PancakeSwap pair contract, many functions fail if there are no tokens actually transferred. If no tokens are sent, the `swapTokensForEth` and `addLiquidity` calls to PancakeSwap and might thus revert the transaction.
Recommendation(s)	Consider wrapping the call to `swapTokensForEth` in safety functions:

```
if (half > 0) {
  swapTokensForEth(half);
}

uint256 newBalance = address(this).balance.sub(initialBalance);

if (newBalance > 0) {
  addLiquidity(otherHalf, newBalance);
}
```

Resolution	RESOLVED The recommended safety functions have been implemented.

Issue #21	`delegateBySig` can be frontrun and cause denial of service
Severity	INFORMATIONAL
Description	Currently if delegateBySig is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `delegateBySig` transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example.
Recommendation(s)	If breaking with EIP-2612 compliancy is not a problem, consider adding the desired message sender in the struct hash and requiring this desired sender to be equal to `msg.sender`. This reduces the problem to having the message sender be able to frontrun you which is okay if it is a reviewed contract. A second possibility, which would keep compliance with EIP-2612 is to fail silently in the executing contract using a try structure introduced in Solidity 0.6, or only doing the `delegateBySig` call if the delegation is currently not done.
Resolution	ACKNOWLEDGED Client has confirmed that since this does not affect any of the core business logic, they are not as concerned about this issue. The delegate function will only be used for voting on proposals.

Issue #22	Deployer account is excluded from tax
Severity	INFORMATIONAL
Description	During deployment, the contract creator is automatically excluded from the transfer tax.
Recommendation(s)	Consider including this account again to increase user confidence in the project.
Resolution	RESOLVED The client has said that this account is only excluded for the initial liquidity generation and that this account will be included again afterwards.

Issue #23	Lack of events for `includeInTax` and `excludeFromTax`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation(s)	Add events for `includeInTax` and `excludeFromTax`. Consider also emitting events in the constructor when `owner()` and `address(this)` are excluded.
Resolution	RESOLVED The recommended events have been added.

Issue #24	`includeInTax`, `excludeFromTax`, `transferOperator`, `updateGoofySwapRouter`, `updateSwapAndLiquifyEnabled`, `updateMinAmountToLiquify`, `updateBurnRate` **and** `updateTransferTaxRate` **can be made** `external`
Severity	INFORMATIONAL
Description	The `includeInTax`, `excludeFromTax`, `transferOperator`, `updateGoofySwapRouter`, `updateSwapAndLiquifyEnabled`, `updateMinAmountToLiquify`, `updateBurnRate` and `updateTransferTaxRate` 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 (https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices).
Recommendation(s)	Consider making these functions `external`.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY