PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Hugin Finance

24 August 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 Hugin Finance 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	Hugin Finance
URL	https://huginfinance.com/
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

The contracts were provided to Paladin in a GitHub repository. Once the project has deployed the final contracts, we will update this report after verifying that the deployed contracts match the contracts we have audited.

Name	Contract	Live Code Match
`HuginToken`	`HuginToken.sol`	
`MasterChef`	`MasterChef.sol`	
`Timelock`	`Timelock.sol`	

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1	-	-
Medium	2	2	-	-
Low	5	3	-	2
Informational	8	8	-	-
Total	**16**	**14**	**-**	**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 HuginToken

ID	Severity	Summary	Status
01	MEDIUM	The `updateMaxTransferAmountRate` minimum safeguard is too low	RESOLVED
02	LOW	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	ACKNOWLEDGED
03	LOW	LP tokens are sent to the operator address which could be an EOA	ACKNOWLEDGED
04	INFORMATIONAL	Exclusion of zero address from anti-whale in constructor is redundant with the ERC-20 library	RESOLVED
05	INFORMATIONAL	Exclude relevant contracts from anti-whale	RESOLVED
06	INFORMATIONAL	delegateBySig can be frontrun and cause denial of service	RESOLVED
07	INFORMATIONAL	`mint`, `updateTransferTaxRate`, `updateBurnRate`, `updateMaxTransferAmountRate`, `updateMinAmountToLiquify`, `setExcludedFromAntiWhale`, `updateSwapAndLiquifyEnabled`, `transferOperator` can be made `external`	RESOLVED

1.3.2 MasterChef

ID	Severity	Summary	Status
08	HIGH	Severely excessive rewards issue when a token with a transfer tax is added	RESOLVED
09	MEDIUM	Setting `feeAddress` to the zero address will break most functionality	RESOLVED
10	LOW	Adding an EOA or a non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate`	RESOLVED
11	LOW	Pools use the contract balance to figure out the total deposits	RESOLVED
12	LOW	The `pendingHgn` function will revert if `totalAllocPoint` is zero	RESOLVED
13	INFORMATIONAL	Token hardcap can be slightly breached	RESOLVED
14	INFORMATIONAL	`_referrer` does not have to be cast to an address	RESOLVED
15	INFORMATIONAL	Lack of events for `add` and `set`	RESOLVED
16	INFORMATIONAL	`deposit`, `withdraw`, `emergencyWithdraw` functions can be made `external`	RESOLVED

1.3.3 Timelock

No issues found.

2 Findings

2.1 HuginToken

The Hugin token is a simple ERC20 token with governance. It contains an anti-whale transfer limit and a 6% transfer-tax (can be set to a maximum of 10%), of which one-fifth is to be burned – this burn ratio can be set to any value between 0 and 100%. The token operator can exclude addresses from the anti-whale limit. By default, the contract creator is excluded from anti-whale.

2.1.1 Token Overview

Address	To be deployed
Token Supply	5,000,000 (five million)
Decimal Places	18
Transfer Max Size	100%
Transfer Min Size	More than 0%
Transfer Fees	Currently 6%, can be set up to 10%

2.1.2 Privileged Roles

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

- `mint`
- `updateTransferTaxRate`
- `updateBurnRate`
- `updateMaxTransferAmountRate`
- `updateMinAmountToLiquify`
- `setExcludedFromAntiWhale`
- `updateSwapAndLiquifyEnabled`
- `transferOperator`

2.1.3 Issues & Recommendations

Issue #01	The `updateMaxTransferAmountRate` minimum safeguard is too low
Severity	MEDIUM SEVERITY
Description	The purpose of anti-whales is to prevent large amounts of selling quickly. However, setting the minimum to only 0.1% allows the project to hamper users' efforts in trying to liquidate their positions in times of need or haste. Additionally, setting it so low only marginally delays the selling pressure, and would likely cause further chaos and anger should there be any marked attempt to prevent users from exercising their freedom to liquidate.
Recommendation	Consider increasing the minimum to at least 1-2%, which would be a more tolerable amount for many users. `require(_maxTransferAmountRate > 200, "HGN::updateMaxTransferAmountRate: Invalid amount");`
Resolution	RESOLVED The minimum anti-whale is now 2%.

Issue #02	**`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 tokens for dumping. As token ownership remains with the deployer before ownership is transferred to the Masterchef, this `mint` function may be called for an arbitrary number of reasons.
Recommendation	Consider being forthright if this `mint` function has been 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	ACKNOWLEDGED 1 million tokens are pre-minted to the owner on deployment. Once token ownership has been transferred to the Masterchef, we shall mark this issue as Resolved.

Issue #03	LP tokens are sent to the operator address which could be an EOA
Severity	LOW SEVERITY
Description	A part of the transaction fee is used to generate LP tokens. These are sent to the operator address, which is normally an EOA. As a result, these may be offloaded or 'dumped' on users.
Recommendation	Consider sending the LP tokens to a vesting contract which will release the tokens gradually over time. Alternatively, sending them to the burn address may be feasible and may be looked upon favourably by users.
Resolution	ACKNOWLEDGED

Issue #04	Exclusion of zero address from anti-whale in constructor is redundant with the ERC-20 library
Severity	INFORMATIONAL
Description	Within the HuginToken constructor, the zero address is excluded from the anti-whale limit. However, no transactions can be made to the zero address since they are reverted in the underlying ERC-20 library.
Recommendation	Consider removing the unnecessary exclusion.
Resolution	RESOLVED

Issue #05	Exclude relevant contracts from anti-whale
Severity	INFORMATIONAL
Description	Currently, the Masterchef contract has already been excluded from anti-whale. In the future, should there be a redeployment, the project must ensure that the Masterchef contract is excluded from anti-whale as well, as not doing so will cause harvesting to revert in the Masterchef should the amount exceed the anti-whale limitation.
Recommendation	No resolutions required.
Resolution	RESOLVED

Issue #06	`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	Consider taking this behavior into consideration in derivative contracts that use this function. These derivative contracts could for example only execute `delegateBySig` if the current delegation is not yet set correctly. This way the frontrunning will have no effect.
Resolution	RESOLVED `delegateBySig` has been removed.

Issue #07	**`mint`, `updateTransferTaxRate`, `updateBurnRate`, `updateMaxTransferAmountRate`, `updateMinAmountToLiquify`, `setExcludedFromAntiWhale`, `updateSwapAndLiquifyEnabled`, `transferOperator` can be made `external`**
Severity	INFORMATIONAL
Description	The above 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

2.2 MasterChef

The Masterchef is a fork of Goose Finance's Masterchef, with the addition of referral recording and commission features. A notable feature of forking this Masterchef is the removal of the migrator function from Pancakeswap, which of late has been used maliciously to steal users' tokens. We commend Hugin Finance on their decision to fork a relatively safer version of the Masterchef.

There is an upper limit of 4% for deposit fees, and emissions too have an upper limit of 2 Hugin tokens per block.

2.2.1 Privileged Roles

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

- `add`
- `set`
- `setFeeAddress`
- `updateEmissionRate`
- `setReferralCommissionRate`
- `payReferralCommission`

2.2.2 Issues & Recommendations

Issue #08	**Severely excessive rewards issue when a token with a transfer tax is added**
Severity	HIGH SEVERITY
Description	When tokens with a transfer tax are added to the pools, this will result in significant excessive rewards. Due to the way the Masterchef handles rewards, rewards can be heavily inflated when the balance of the Masterchef no longer matches that of user deposits. This happens for example with transfer tax tokens. This issue is further amplified on Masterchefs like this one with a referral mechanism, since tokens can be minted directly. This flaw of the Masterchef has recently been exploited on a significant number of projects, all of which their native tokens went to $0 afterwards because the exploit resulted in a large number of native tokens being minted and dumped. This issue was also present in SushiSwap (the original Masterchef). Since they were never meant to have any tokens but LP tokens, it was not a problem there but has become a problem to projects who have started forking it for usage with less standard tokens.

Recommendation	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:
	
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.transferFrom(msg.sender, address(this), _amount);`
	`_amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`
	
	Note that by using this method, you can also add the specific transfer tax logic for the native token if you so wish.
	
	In the short term, as the contract is not upgradeable, consider carefully choosing the tokens to be added to the Masterchef. As long as a token with a transfer tax (or other mechanism that can affect the balances) is not added to the pools, this issue should not present itself.

Resolution	RESOLVED However, the placement of `setReferral` in the before-after method is highly unusual.

Issue #09	**Setting `feeAddress` to the zero address will break most functionality**
Severity	MEDIUM SEVERITY
Description	Within the token contract, minting or transferring tokens to the zero address will revert the transaction. Deposits and withdrawals will break if the `feeAddress` is ever set to the zero address. Harvesting will fail as well
Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like `require(_feeAddress != address(0), "!nonzero");` to the configuration function.
Resolution	RESOLVED

Issue #10	**Adding an EOA or a non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate`**
Severity	LOW SEVERITY
Description	`updateEmissionRate` will always call `balanceOf(address(this))` on the token of this pool, and will fail if the token is not an actual token contract address.
Recommendation	Consider simply adding a test line in the add function. If the token does not exist, this will make sure the add function fails. `_lpToken.balanceOf(address(this));`
Resolution	RESOLVED

Issue #11	Pools use the contract balance to figure out the total deposits
Severity	LOW SEVERITY
Location	Line 1262 `uint256 lpSupply = pool.lpToken.balanceOf(address(this));` Line 1287 `uint256 lpSupply = pool.lpToken.balanceOf(address(this));`
Description	The total number of tokens in the Masterchef contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the Masterchef. More severely, because the native token is constantly minted, this will cause severe dilution on the native token pool.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits, rather than calculating `lpSupply` based on the token balance in the Masterchef. Note that this `pool.lpSupply` would then be used to more accurately calculate deposits, withdrawals, and rewards.
Resolution	RESOLVED

Issue #12	The `pendingHgn` function will revert if `totalAllocPoint` is zero
Severity	LOW SEVERITY
Description	In the `pendingHgn` function, at some point a division is made by the `totalAllocPoint` variable. If all pools have their rewards set to zero, this variable will be zero as well. The requests will then revert with a division by zero error.
Recommendation	Consider only calculating the accumulated rewards since the `lastRewardBlock` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.number` and `lpSupply`, like so: `if (block.number > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	RESOLVED

Issue #13	Token hardcap can be slightly breached
Severity	INFORMATIONAL
Description	Currently, the hard-cap of 5 million tokens is enforced through the reward multiplier as follows: `if (huginToken.totalSupply() >= MAX_hugin_SUPPLY) return 0;` However, this only checks that the limit is not reached before the mint. This could mean that for example if the current supply is 4,999,999 tokens and there's a request to mint 2 tokens, this request will still pass since the supply is not reached and the final supply will be 5,000,001 tokens.

Recommendation	Consider not minting the tokens in case the supply exceeds the total supply. The least intrusive solution is to change
	
	`huginToken.mint(devAddress, huginReward.div(10));`
	`huginToken.mint(address(this), huginReward);`
	
	to
	
	`if (huginToken.totalSupply().add(huginReward.mul(11).div(10)) <= MAX_hugin_SUPPLY) {`
	`  // The whole emission can be mint`
	`  huginToken.mint(devAddress, huginReward.div(10));`
	`  huginToken.mint(address(this), huginReward);`
	`} else if (huginToken.totalSupply() < MAX_hugin_SUPPLY) {`
	`  // The emission can be partially mint`
	`  huginToken.mint(address(this), MAX_hugin_SUPPLY.sub(huginToken.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 The client has implemented the least intrusive solution to ensure only the max token supply is minted.

Issue #14	`_referrer` does not have to be cast to an address
Severity	INFORMATIONAL
Description	In the `setReferral` function, `_referrer` is cast to address, which is not necessary as it is already an address.
Recommendation	Consider instead using the following if statement, which includes a check for non-zero address, and is the standard by which Panther referrals are done: `if (_user != address(0) && _referrer != address(0) && _user != _referrer && referrers[_user] == address(0))`
Resolution	RESOLVED

Issue #15	Lack of events for add and set
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for `add` and `set`.
Resolution	RESOLVED

Issue #16	`deposit`, `withdraw`, `emergencyWithdraw` functions can be made `external`
Severity	INFORMATIONAL
Description	The above 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

2.3 Timelock

The Timelock contract is a clean fork of Compounder 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	TBD	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	6 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.
Maximum Delay	30 days	The maximum delay indicates the highest value that the delay can be set.
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.

2.3.1 Issues & Recommendations

No issues found


PALADIN
BLOCKCHAIN SECURITY