PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Hermes Defi

11 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 Hermes Defi. 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	Hermes Defi
URL	https://www.hermesdefi.io/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

We have verified that the deployed contracts below match the final contracts that we have audited. Users are advised to check that they are interacting with the contracts listed here.

Name	Contract	Live Code Match
IrisToken	0xdaB35042e63E93Cc8556c9bAE482E5415B5Ac4B1	MATCH
MasterChef	0x4aA8DeF481d19564596754CD2108086Cf0bDc71B	MATCH
Referral	0xDC99FE88118CdE8316df10Eb16c722C3967e73Fd	MATCH
Timelock	0xe83ECF8077538AD80373e7Cc12FDfeffA3B63559	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1	-	-
Medium	1	1	-	-
Low	6	6	-	-
Informational	8	8	-	-
Total	**16**	**16**	**0**	**0**

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 IrisToken

ID	Severity	Summary	Status
01	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	RESOLVED
02	INFORMATIONAL	Governance functionality is broken (present in all Goose forks)	RESOLVED
03	INFORMATIONAL	`delegateBySig` can be frontrun and cause denial of service	RESOLVED

1.3.2 MasterChef

ID	Severity	Summary	Status
04	HIGH	Severely excessive rewards issue when a token with a transfer tax is added	RESOLVED
05	MEDIUM	`updateStartBlock` does not update `poolInfo.lastRewardBlock`	RESOLVED
06	LOW	Setting `devAddress` and `feeAddress` to the zero address will break `updatePool` and `deposit` functionality respectively	RESOLVED
07	LOW	Adding an EOA or non-token contract as a pool will break updatePool, massUpdatePools and updateEmissionRate	RESOLVED
08	LOW	The pendingIris function will revert if totalAllocPoint is zero	RESOLVED
09	LOW	updateEmissionRate has no maximum safeguard	RESOLVED
10	INFORMATIONAL	Pools use the contract balance to figure out the total deposits	RESOLVED
11	INFORMATIONAL	Native tokens can be minted beyond the hard-cap limitation	RESOLVED
12	INFORMATIONAL	There are no sanity checks in the `setReferralAddress` function	RESOLVED
13	INFORMATIONAL	`deposit`, `withdraw`, `emergencyWithdraw`, and `updateStartBlock` functions can be made external	RESOLVED
14	INFORMATIONAL	Lack of events for `add`, `set`, `setReferralCommissionRate`, and `updateStartBlock`	RESOLVED

1.3.3 Referral

ID	Severity	Summary	Status
15	LOW	The owner of the Referral contract can overwrite themselves as the referrer for all users using the `recordReferral` function	RESOLVED
16	INFORMATIONAL	recordReferral function can be made external	RESOLVED

1.3.4 Timelock

No issues found.

2 Findings

2.1 IrisToken

2.1.1 Token Overview

Address	0x15a1DF7ffda9F4921A187DB2c530902ba955c64D
Token Supply	1,000,000 (one million) - enforced in the Masterchef
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None

2.1.2 Privileged Roles

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

- `renounceOwnership`
- `transferOwnership`
- `mint`

2.1.3 Issues & Recommendations

Issue #01	`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 has already been transferred to the Masterchef, the main risk that remains is whether any tokens were pre-minted to the project owner prior to transferring ownership.
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	RESOLVED Hermes has described their planned token distributions clearly in their documentation. - https://hermes-defi.gitbook.io/hermes-finance/tokenomics-1/token - https://hermes-defi.gitbook.io/hermes-finance/launch/pre-sale-fenix-2

Issue #02	Governance functionality is broken (present in all Goose forks)
Severity	INFORMATIONAL
Description	Although there is YAM-related delegation code in the token contract which is usually used for governance and voting, the delegation code can be abused as the delegates are not moved during transfers and burns. This allows for double spending attacks on the voting mechanism. It should be noted that this issue is present in pretty much every single farm out there including PancakeSwap and even SushiSwap.
Recommendation	The broken delegation-related code can be removed to reduce the size of the contract. If voting is ever desired, it can still be done through snapshot.org, which is used by many of the larger projects.
Resolution	RESOLVED All governance functionality was removed.

Issue #03	`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	As recommended in the previous issue, the delegation-related code can just be removed.
Resolution	RESOLVED All governance functionality was removed.

2.2 MasterChef

The Masterchef is a fork of Goose Finance's Masterchef. 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. Furthermore, in comparison to Goose Finance, Hermes Finance has limited the deposit fee to at most 5%. We commend Hermes Finance on their decision to fork a relatively safer version of the Masterchef and trim down the governance privileges with regards to the deposit fees.

The main other extension this contract has is a referral mechanism that currently gives the referrer a 2% referral commission of earned rewards (up to 5% maximum).

2.2.1 Privileged Roles

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

- `setReferralCommissionRate`
- `setReferralAddress`
- `updateEmissionRate`
- `updateStartBlock`
- `setFeeAddress`
- `setDevAddress`
- `renounceOwnership`
- `transferOwnership`
- `add`
- `set`ssues & Recommendations

Issue #04	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);
```

Resolution	RESOLVED The recommended before-after check has been implemented.

Issue #05	`updateStartBlock` does not update `poolInfo.lastRewardBlock`
Severity	MEDIUM SEVERITY
Description	When users deposit in a pool, they will start earning rewards as soon as the current block is larger than the pools' `lastRewardBlock`. When a pool is added, this `lastRewardBlock` is set to either the `currentBlock` or the `startBlock` (whichever is greater). If the `startBlock` is moved, the `lastRewardBlock` absolutely has to be moved as well. There have been multiple cases of projects that have failed to start as their `lastRewardBlock` was still weeks in the future, even though they called `updateStartBlock`. This issue can cause significant reputational damage.

Recommendation	In the long term, consider adding a loop to update all `lastRewardBlocks` in the `updateStartBlock` function. Note that this loop may run out of gas if a very significant amount of pools are added.
	
	`function updateStartBlock(uint256 _startBlock) external onlyOwner`
	`{`
	`  require(startBlock > block.number, "Farm already started");`
	`uint256 length = poolInfo.length;`
	`for (uint256 pid = 0; pid < length; ++pid) {`
	`  PoolInfo storage pool = poolInfo[pid];`
	`  pool.lastRewardBlock = _startBlock;`
	`}`
	`startBlock = _startBlock;`
	`}`
	
	In the short term, consider not calling this function anymore as it will not update the pools.

Resolution	RESOLVED The recommended code block has been added.

Issue #06	**Setting `devAddress` and `feeAddress` to the zero address will break `updatePool` and `deposit` functionality respectively**
Severity	LOW SEVERITY
Description	The `updatePool` function will mint tokens to the `devAddress`, but minting will revert if it is made to the zero address. Transferring tokens to the zero address will revert the transactions as well. Deposits will thus break if the `feeAddress` is ever set to the zero address. Deposit-based harvests will break as well.

Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like
	`require(_devAddress != address(0), "!nonzero");`
	to the `setDevAddress` function, and
	`require(_feeAddress != address(0), "!nonzero");`
	to the `setFeeAddress` function.

Resolution	RESOLVED The recommended requirements have been added.

Issue #07	**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 the 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 The recommended check has been added at the start of the add function.

Issue #08	**The `pendingIris` function will revert if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	In the `pendingIris` 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 The recommended check has been added to the `if` clause.

Issue #09	`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(_irisPerBlock <= MAX_EMISSION_RATE,"Too high");`
Resolution	RESOLVED The recommended requirement has been added with a maximum of 1 token per block.

Issue #10	Pools use the contract balance to figure out the total deposits
Severity	INFORMATIONAL
Description	As with pretty much all Masterchefs, 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. Each `lpToken.balanceOf(address(this))` query can then be replaced with this `lpSupply` as well.
Resolution	RESOLVED There is now an `lpSupply` variable that correctly keeps track of the pool balances and is used everywhere where the total supply is needed in the code.

Issue #11	Native tokens can be minted beyond the hard-cap limitation
Severity	INFORMATIONAL
Description	Currently, the hard-cap of 1,000,000 tokens is enforced through the reward multiplier as follows: `if (iris.totalSupply() >= max_iris_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 999,999 tokens and there's a request to mint 2 tokens, this request will still pass since the maximum supply has not been reached and thus the final supply will be 1,000,001 tokens.

Recommendation	Consider not minting the tokens in case the supply exceeds the total supply. The least intrusive solution is to change:
	`iris.mint(devAddress, irisReward.div(20));`
	`iris.mint(address(this), irisReward);`
	to
	`if (iris.totalSupply().add(irisReward.mul(105).div(100)) <= max_iris_supply) {`
	`    // The whole emission can be mint`
	`    iris.mint(devAddress, irisReward.div(20));`
	`    iris.mint(address(this), irisReward);`
	`} else if (iris.totalSupply() < max_iris_supply) {`
	`    // The emission can be partially mint`
	`    iris.mint(address(this), max_iris_supply.sub(iris.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 recommended code block has been implemented correctly in both `updatePool` and the referral minting section.

Issue #12	There are no sanity checks in the `setReferralAddress` function
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, such as setting it in the constructor. We rarely ever see a project updating their referral after it is initially set.
Resolution	RESOLVED The referral address can be set only once now.

Issue #13	`deposit`, `withdraw`, `emergencyWithdraw`, and `updateStartBlock` functions can be made `external`
Severity	INFORMATIONAL
Description	`deposit`, `withdraw`, `emergencyWithdraw`, and `updateStartBlock` 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 #14	**Lack of events for `add`, `set`, `setReferralCommissionRate`, and `updateStartBlock`**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

2.3 Referral

The Referral contract is a clean fork of the Pantherswap Referral contract, without the ability to transfer out tokens sent there by mistake.

2.3.1 Privileged Roles

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

- `recordReferral`
- `updateOperator`
- `renounceOwnership`
- `transferOwnership`

2.3.2 Issues & Recommendations

Issue #15	**The owner of the Referral contract can overwrite themselves as the referrer for all users using the `recordReferral` function**
Severity	LOW SEVERITY
Description	The operator should only be the Masterchef contract. However, the owner of the Referral contract has privileges that may be abused. The following steps detail how the owner can make themselves the referrer for all users : 1. Owner of the Referral contract calls updateOperator to add themselves as an operator. 2. As an operator, they then call recordReferral with their own wallet address as the referrer for each user. 3. This results in their wallet address being minted potentially large amounts of referral commission.
Recommendation	There are 3 possible recommendations : 1. Consider making the `updateOperator` function callable only once. 2. Setting only the Masterchef as an operator in the modifier, and removing the `updateOperator` function. 3. Calling the `updateOperator` to set the Masterchef as operator, then renouncing ownership of the Referral contract such that it cannot be called in the future.
Resolution	RESOLVED The operator can now only be updated once.

Issue #16	`recordReferral` function can be made `external`
Severity	INFORMATIONAL
Description	The `recordReferral` function 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 this function `external`.
Resolution	RESOLVED

2.4 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	12 hours	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	12 hours	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

2.4.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY