PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Delirium
by Sandman Finance

11 September 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 the Delirium layer of Sandman Finance on the Polygon network. 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	Delirium by Sandman Finance
URL	https://delirium.sandman.finance/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
DelightPresaleVIP	0x5742477df10C1cC973e1B9A6542d588BfF093D99	MATCH
DelightToken	0x960fE3CBEF6d922eCe9fa3E5611B8f9Ec14DE649	MATCH
DelightMadness	0x377a7C1d28aCd4eca2870128645475b9b27A399f	MATCH
DeliriumToken	0x238779aFfE6FFD475cB7e84582FcA7789F310Dc8	MATCH
MasterChef	0x9D73BEFA87A51562464d07fE5959476f64Cfdc7f	MATCH
Timelock	0xc86Cc8d4AA89487B1f1166E90E6E5d42182ED38d	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	0	-	-	-
Low	7	6	1	-
Informational	2	2	-	-
Total	**9**	**8**	**1**	**-**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 DelightPresaleVIP

ID	Severity	Summary	Status
01	LOW	Sandman tokens should be burned	RESOLVED
02	LOW	`endBlock` has no safeguards in constructor	RESOLVED

1.3.2 DelightPresaleVIP

ID	Severity	Summary	Status
03	LOW	`mintDelightForSandManL1` function could arbitrarily mint large amounts of presale tokens	RESOLVED

1.3.3 DelightMadness

No issues found.

1.3.4 DeliriumToken

ID	Severity	Summary	Status
04	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	PARTIAL

1.3.5 MasterChef

ID	Severity	Summary	Status
05	LOW	`setStartBlock` does not update each pool's `lastRewardBlock`	RESOLVED
06	LOW	`setEmissionRate` has no maximum safeguard	RESOLVED
07	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED
08	INFO	`setEmissionRate` can be made `external`	RESOLVED

1.3.6 Timelock

No issues found.

2 Findings

2.1 DelightPresaleVIP

The DelightPresaleVIP contract allows for Sandman token holders to swap their tokens for the presale Delight tokens at a 1:0.48 rate – that is, 1 Sandman token can be redeemed for 0.48 Delight tokens. These Sandman tokens would be sent to the `feeAddress`.

2.1.1 Privileged Roles

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

- `setStartBlock`
- `setSaleVipPrice`
- `setDelightVipPresaleSize`
- `sendUnclaimedDelightToDeadAddress`

2.1.2 Issues & Recommendations

Issue #01	Sandman tokens should be burned
Severity	LOW SEVERITY
Description	Presumably, the first layer Sandman would stop emissions once the supply has reached 100,000 tokens, and thus support would stop for the native tokens of that layer. As there should be no market remaining (all Sandman token holders should withdraw their LP and get their tokens ready for swapping to Delight), these Sandman tokens should be burned rather than sent to the `feeAddress`, which could dump the tokens on any users who have yet to withdraw their Sandman LP tokens (either they forgot, or are delayed in doing so), essentially nuking the price.
Recommendation	In the `swapSandManForDelight` function, consider sending the Sandman tokens to the burn address instead, like so: `require(IERC20(sandManAddress).transferFrom(msg.sender, BURN_ADDRESS, sandManToSwap), "E18");`
Resolution	RESOLVED The client resolves to manually burn all the tokens in the Fee Wallet. This can be manually verified at https://polygonscan.com/address/0xFbd37BfD354611b809054CBA049AAa7CC5fD364b

Issue #02	endBlock has no safeguards in constructor
Severity	LOW SEVERITY

Description	If for whatever reason the `endBlock` is accidentally set to a number below `startBlock`, then the `swapSandManForDelight` function would never be able to function and thus users would not be able to swap their Layer1 tokens to the Layer2 presale tokens, due to the following lines:
	`require(block.number >= startBlock, "E10");`
	`require(block.number < endBlock, "E11");`

Recommendation(s)	Consider adding in a simple safeguard in the constructor, like so: `require(_endBlock > _startBlock, “endBlock is in the past!”);`
Resolution	RESOLVED The contract has already been deployed and operational, and `endBlock` is set beyond `startBlock`.

2.2 DelightToken

The DelightToken presale contract is a fork of MorpheusToken in the Sandman Layer (which in turn is a fork of the PolyWantsACracker's presale contract). Each presale token is priced at 10 USDC, with a maximum of 500 tokens purchasable per wallet. The presale duration lasts for at least 3 days, though this can be extended if desired (as block times are non-constant on Polygon, this duration may fluctuate).

75,000 presale tokens are minted for public sale, and a further 48,000 presale tokens are minted for the VIP presale (those swapping from Sandman tokens). As a result, there would be 123,000 presale tokens total in circulation. This is subject to change should the client wish to adjust their presale price and allocation size.

2.2.1 Privileged Roles

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

- `setStartBlock`
- `mintDelightForSandManL1`

2.2.2 Issues & Recommendations

Issue #03	**`mintDelightForSandManL1` function could arbitrarily mint large amounts of presale tokens**
Severity	LOW SEVERITY
Description	The `mintDelightForSandManL1` function can be called by the owner at any time to print an infinite number of presale tokens to any address (though it can be called only once), which can then be swapped over to the native Delirium token via the DelightMadness contract. Although we believe this event will be highly unlikely, it is nonetheless a possibility.
Recommendation	Consider minting the required number of tokens to the `presaleVipAddress` (which we believe is the DelightPresaleVIP contract) in the constructor and removing this function. That way, users would know for certain the number of tokens minted and the recipient address. Of course, should the client wish to change their presale price and allocation size, then removing this function may not be ideal. In that event, we may be able to mark this issue as Resolved if the client acknowledges this and commits to ensuring that they are transparent as to how many presale tokens are minted when this function is called, and that it should match the desired allocation size to ensure that all Sandman tokens can be swapped without issue.
Resolution	RESOLVED Only 123,000 presale tokens have been minted for sale. The function can no longer be called.

2.3 DelightMadness

The DelightMadness contract is a perfect fork of MorpheusDream in the Sandman layer (which is a fork of the PolyWantsACracker's swap contract). The contract swaps presale Delight tokens for the native Delirium token at a 1:1 rate. The Delight tokens are burned, while any excess (unredeemed) Delirium tokens may also be burned by the Owner. In order for the swaps to take place, sufficient Delirium tokens have to be transferred to this swap contract.

2.3.1 Privileged Roles

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

- `sendUnclaimedDeliriumToDeadAddress`
- `setStartBlock`

2.3.2 Issues & Recommendations

No issues found.

2.4 DeliriumToken

This is a simple ERC-20 contract with the broken delegation code removed. This looks to be the native token that will be used in this layer. As there are 123,000 presale tokens, the client must ensure that 123,000 Delirium tokens are pre-minted and transferred to the DelightMadness contract to ensure that users would be able to successfully swap their tokens.

2.4.1 Token Overview

Address	0x238779aFfE6FFD475cB7e84582FcA7789F310Dc8
Token Supply	500,000 (five hundred thousand)
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None
Pre-mints	TBC

2.4.2 Privileged Roles

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

- `mint`

2.4.3 Issues & Recommendations

Issue #04	`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.
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	PARTIALLY RESOLVED After the presale has ended and sufficient tokens have been minted for that purpose, the client shall transfer ownership of the token to the Masterchef. Once that has occured, we shall mark this issue as Resolved.

2.5 MasterChef

This Masterchef is forked from its previous layer, Sandman, which itself was forked from PolyWantsACracker. Both have been thoroughly audited by Paladin, and thus lacks any obvious malicious vectors which may pose a threat to users' funds.

2.5.1 Privileged Roles

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

- `add`
- `set`
- `setFeeAddress`
- `setStartBlock`
- `setEmissionRate`

2.5.2 Issues & Recommendations

Issue #05	**`setStartBlock` does not update each pool's `lastRewardBlock`**
Severity	LOW SEVERITY
Description	The `setStartBlock` function can be called even after pools have been added. Unfortunately, when called, this function merely updates the global `startBlock` without updating each pool's `lastRewardBlock`. This may result in emissions not starting on the expected block.

Recommendation	Consider implementing the following for `setStartBlock`:
	
	`function setStartBlock(uint256 _newStartBlock) external onlyOwner {`
	`  require(block.number < startBlock, "cannot change start block if farm has already started");`
	`  require(block.number < _newStartBlock,"cannot set start block in the past");`
	`  uint256 length = poolInfo.length;`
	`  for (uint256 pid = 0; pid < length; ++pid) {`
	`  PoolInfo storage pool = poolInfo[pid];`
	`  pool.lastRewardBlock = _newStartBlock;`
	`  }`
	`  startBlock = _newStartBlock;`
	`  emit UpdateStartBlock(startBlock);`
	`}`
	
	Note that this loop may run out of gas if a very significant amount of pools are added.

Resolution	RESOLVED The function can only be called before pools have been added to the Masterchef.

Issue #06	**`setEmissionRate` 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(_deliriumPerBlock <= MAX_EMISSION_RATE,"Too high");`
Resolution	RESOLVED There is now a maximum of 10 tokens per block.

Issue #07	**Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `_updatePool`, `pool.accDeliriumPerShare` is based on the `pool.lpSupply` variable. `pool.accDeliriumPerShare = pool.accDeliriumPerShare + ((deliriumReward * 1e12) / pool.lpSupply);` However, if this `pool.amount` becomes a severely large value this will cause precision errors due to rounding. This is famously seen when pools decide to add meme tokens which usually have a huge token supply and no decimals.
Recommendation	Consider increasing precision to `1e18` across the entire contract.
Resolution	RESOLVED

Issue #08	`setEmissionRate` can be made `external`
Severity	INFORMATIONAL
Description	This 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

2.6 Timelock

The Timelock contract is a clean fork of Compound 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	8 hours	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.
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.


PALADIN
BLOCKCHAIN SECURITY