PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Bulls Inc (Farm)

11 February 2022

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.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Bulls Inc's farm contracts on the Fantom Opera 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	Bulls Inc (Farm contracts)
URL	https://bulls.inc
Platform	Fantom Opera
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`BULLSToken`	`0x6f945C83f9DDc4EBd9Cad21e7C08Cc3c82A21c90`	MATCH
`MasterChef`	`0x04Da1333a9F19321EB9841D421C4528A6b16c31a`	MATCH
`Timelock`	`0xb20D869E1D134026fb32C28B0e246CEB356AECae`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	1	1	-	-
Low	4	4	-	-
Informational	12	7	2	3
Total	**17**	**12**	**2**	**3**

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 BULLSToken [BULLS-INC [BULLS]]

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	INFO	`mint` and `burn` can be made external	RESOLVED
03	INFO	The balance check on `burn` method can be removed	RESOLVED
04	INFO	Unnecessary casting of `msg.sender` into `address`	RESOLVED

1.3.2 MasterChef

ID	Severity	Summary	Status
05	MEDIUM	Adding an address that does not implement the IERC20 interface will cause `massUpdatePools` and `updatePool` to fail	RESOLVED
06	LOW	Lack of non-zero address check in the constructor for fee address	RESOLVED
07	LOW	The `pendingReward` function will revert if `totalAllocPoint` is zero	RESOLVED
08	LOW	`setEmissionRate` does not update pools	RESOLVED
09	INFO	`startBlock` can be set to a time before deployment time in the constructor	RESOLVED
10	INFO	Pools use the contract balance to figure out the total deposits	ACKNOWLEDGED
11	INFO	Unnecessary casting of `msg.sender` into `address`	RESOLVED
12	INFO	Rounding issue due to tokens with a very large supply can cause large supply tokens to receive zero emissions	ACKNOWLEDGED
13	INFO	Unnecessary assignment to 0	PARTIAL
14	INFO	Unused variables/events	PARTIAL
15	INFO	Lack of events for several functions	ACKNOWLEDGED
16	INFO	`MAX_EMISSION_RATE` can be declared as a constant	RESOLVED
17	INFO	`farmToken` can be made immutable	RESOLVED

1.3.3 Timelock

No issues found.

2 Findings

2.1 BULLSToken [BULLS-INC [BULLS]]

The BULLS token is an ERC-20 token which will be used as the main reward token for the Masterchef. It allows for BULLS tokens to be minted when the `mint` function is called by the owner of the contract and burned by whoever holds any amount of BULLS.

The owner of the contract at the time of deployment would be the Bulls Inc team. The ownership of the contract should be transferred to the Masterchef after deployment.

2.1.1 Token Overview

Address	`0x6f945C83f9DDc4EBd9Cad21e7C08Cc3c82A21c90`
Token Supply	Unlimited
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	1 [to the deployer]

2.1.2 Privileges

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

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

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 and dump tokens when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this `mint` function is to be 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 The ownership has been transferred to the Masterchef, 1 BULLS has been minted for the LP.

Issue #02	`mint` and `burn` can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. 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 marking the above functions as external.
Resolution	RESOLVED

Issue #03	The balance check on `burn` method can be removed
Severity	INFORMATIONAL
Description	The `burn` method is used to burn a token by the holder doing a check if the balance of the `msg.sender` is >= than the amount. This check is not required as the `_burn` method will revert with underflow if the `msg.sender` does not have enough balance.
Recommendation	Consider removing the `require` check.
Resolution	RESOLVED

Issue #04	Unnecessary casting of `msg.sender` into `address`
Severity	INFORMATIONAL
Location	Line 690 `_mint(address(msg.sender), 1 ether);`
Description	The casting of `msg.sender` to `address` is unnecessary as it is already an `address` datatype.
Recommendation	Remove the casting of `msg.sender` as an `address`.
Resolution	RESOLVED

2.2 MasterChef

The Bulls Inc Masterchef is a fork of Goose Finance's Masterchef. A notable feature of forking this Masterchef is the removal of the `migrator` function from SushiSwap, which can potentially be used to steal user's tokens. Bulls Inc has limited the deposit fee to at most 4%.

The maximum emission that can be set is 5 tokens per block with an initial emission rate of 1.

2.2.1 Privileges

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

- `add`
- `set`
- `transferOwnership`
- `renounceOwnership`
- `setFeeAddress`
- `updateStartBlock`
- `setEmissionRate`

2.2.2 Issues & Recommendations

Issue #05	**Adding an address that does not implement the IERC20 interface will cause `massUpdatePools` and `updatePool` to fail**
Severity	MEDIUM SEVERITY
Description	If the owner adds an address that does not have the IERC20 interface implemented, any functions that call the pool's token address will revert. This would result in `massUpdatePools` being unusable as `updatePool` will be called for the affected pool id and revert. Not being able to call `massUpdatePools` would result in new pool additions affecting existing pool pending rewards as `massUpdatePools` is required to be called each time an addition or change is made to any pool to ensure that existing pools get their fair due rewards before the allocation among pools is changed.
Recommendation	Consider adding a sanity check in the `add` function when adding a token to a new pool: `_lpToken.balanceOf(address(this));`
Resolution	RESOLVED

Issue #06	**Lack of non-zero address check in the constructor for fee address**
Severity	LOW SEVERITY
Description	Although the `feeAddr` is checked to not be the zero address in `setFeeAddress`, it is not done in the constructor. If the contract is deployed with the `feeAddr` set as the zero address, deposits could revert for pools that have a deposit fee if the deposit token does not allow the recipient address of a transfer to be the zero address.
Recommendation	Consider adding the same non-zero address check in the constructor.
Resolution	RESOLVED

Issue #07	**The `pendingReward` function will revert if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	In the `pendingReward` 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.timestamp > pool.lastRewardTime && lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	RESOLVED

Issue #08	**`setEmissionRate` does not update pools**
Severity	LOW SEVERITY
Description	The `setEmissionRate` method used to update the number of tokens emitted each block does not call `massUpdatePools` in order to update the pending rewards, and will thus affect pending rewards. `massUpdatePools` should have a skippable flag like in `add`/`set` (`_withUpdate`).

Recommendation	Consider adding a call to `massUpdatePools` with a flag to skip if necessary.
	`function setEmissionRate(uint256 _emissionRate, bool _withUpdate) external onlyOwner {`
	`  require(_emissionRate <= MAX_EMISSION_RATE, "!max emission rate");`
	`  if (_withUpdate) {`
	`  massUpdatePools();`
	`  }`
	`  tokensPerBlock = _emissionRate;`
	`}`

Resolution	RESOLVED

Issue #09	**`startBlock` can be set to a time before deployment time in the constructor**
Severity	INFORMATIONAL
Description	There is a lack of check that the `startBlock` has to be equal to or after the deployment time
Recommendation	Consider adding a check in the constructor to ensure that the `startBlock` is set at or after the deployment time.
Resolution	RESOLVED

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	ACKNOWLEDGED

Issue #11	Unnecessary casting of `msg.sender` into `address`
Severity	INFORMATIONAL
Location	Line 1233 (Example) `pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);`
Description	There are a number of instances where `msg.sender` is cast into an address, when it is already an `address` datatype.
Recommendation	Remove the casting of `msg.sender` as an `address`.
Resolution	RESOLVED

Issue #12	Rounding issue due to tokens with a very large supply can cause large supply tokens to receive zero emissions
Severity	INFORMATIONAL
Description	Within `updatePool`, `deposit`, `withdraw` and the pending rewards function, `accRewardPerShare` is based on the `lpSupply` variable. `pool.accRewardPerShare = pool.accRewardPerShare.add(reward.mul(1e12).div(lpSupply));` However, if this `lpSupply` becomes a severely large value, this will cause precision errors due to rounding. This is observed when in pools that support tokens with large supplies.
Recommendation	Consider increasing precision to 1e18 across the entire contract.
Resolution	ACKNOWLEDGED

Issue #13	Unnecessary assignment to 0
Severity	INFORMATIONAL
Location	Line 1254 (Example) `uint256 pending = 0;`
Description	The pending variable is automatically assigned to 0 when it is declared as uint256. The assignment to 0 is unnecessary and it consumes gas.
Recommendation	Consider removing the assignment to 0.
Resolution	PARTIALLY RESOLVED

Issue #14	Unused variables/events
Severity	INFORMATIONAL
Description	The following variables/events are not used: - Event `UpdateEmissionRate` - Variable `BONUS_MULTIPLIER` - SafeMath can be removed as the Solidity version is >= 0.8.0 which has built-in SafeMath. - FarmToken contract can be transformed into an interface
Recommendation	Remove unnecessary variables/events.
Resolution	PARTIALLY RESOLVED The client removed the event and converted the FarmToken contract into an interface.

Issue #15	Lack of events for several functions
Severity	INFORMATIONAL
Description	The following functions do not emit an event: - `add` - `set` - `updateStartBlock` - `setEmissionRate` Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	ACKNOWLEDGED

Issue #16	`MAX_EMISSION_RATE` can be declared as a constant
Severity	INFORMATIONAL
Description	Variables that do not change throughout the contract can be marked as constant to signal this to third-party reviewers. This furthermore reduces gas usage.
Recommendation	Consider marking the above variables as constant.
Resolution	RESOLVED

Issue #17	`farmToken` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly immutable.
Resolution	RESOLVED

2.3 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	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.

2.3.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY