PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero (Stargate V2 Fee Claimer)

10 September 2023

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 the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Stargate's v2 fee claimer contracts on the Ethereum, BSC, Avalanche, Polygon, Arbitrum, Optimism and Fantom networks. 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	Stargate (V2 Fee Claimer)
URL	https://layerzero.network/
Platform	Ethereum, BSC, Avalanche, Polygon, Arbitrum, Optimism and Fantom
Language	Solidity
Preliminary Contracts	https://github.com/stargate-protocol/stargate-dao/pull/1/commits/d5558e9e4f04af1883419ffc83a859adab631f63
Resolution 1	https://github.com/stargate-protocol/stargate-dao/tree/7c3b92a674b4b93270e03b35cd465c22d4dbc95c
Resolution 2	https://github.com/stargate-protocol/stargate-dao/tree/3598b574f74fcae5209af233718dcefae5b2d628

1.2 Contracts Assessed

Name	Contract	Live Code Match
`FeeDistributor`	`0xAF667811A7eDcD5B0066CD4cA0da51637DB76D09` (same address on all chains)	MATCH

1.3 Findings Summary

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

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 FeeDistributor

ID	Severity	Summary	Status
01	LOW	Gas griefing: User claims can be made more expensive if a malicious party calls `claimToken` or `claimTokens` on their behalf before they have any VE balance	✓ RESOLVED
02	LOW	Distribution to weeks where no one is supplying to the VE token is completely unclaimable	✓ RESOLVED
03	INFO	Contract is incompatible with multi-address tokens like TUSD	✓ RESOLVED
04	INFO	The uint keyword should not be used	✓ RESOLVED
05	INFO	Additional validation can reduce the state-space for the contract making it more theoretically secure	✓ RESOLVED
06	INFO	Allowing anyone to claim on behalf for other users might be annoying	✓ RESOLVED
07	INFO	Gas optimizations	✓ RESOLVED
08	INFO	Typographical issues	✓ RESOLVED

2 Findings

2.1 FeeDistributor

`FeeDistributor` is a contract which allows for the periodic distribution of any whitelisted token to `veSTG` holders. Usually, protocol owners use this distributor to distribute protocol fees among the holders.

The distribution logic is based on weekly snapshots of `veSTG` holders and tokens are distributed according to their balance of `veSTG`.

To distribute a new token, the team and users can send the tokens to the contract and then call `checkpointToken` before and after the transfer in order to distribute the tokens correctly. The team and users can also call the preferred built-in functions `depositToken`/`depositTokens` which does the same operation automatically while creating appropriately-timed snapshots of the token balance before and after the deposit to ensure that the tokens checkpoint are set in the right epoch.

Fees for any given week are typically distributed to balances from the epoch at the start of that week. These fees become claimable after that week's epoch ends to prevent users claiming fees before all fees have been distributed.

2.1.1 Privileged Functions

- `enableTokenClaiming`
- `withdrawToken`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	**Gas griefing: User claims can be made more expensive if a malicious party calls `claimToken` or `claimTokens` on their behalf before they have any VE balance**
Severity	LOW SEVERITY
Description	Anyone can call `claimToken` and `claimTokens` for any address. These functions also do not fail if that user does not have a VE balance yet or if the token has not yet been deposited into the contract. The latter case where the token has not yet been deposited is not a significant issue since `_userTokenTimeCursor` which is used to index from for that user will simply be set to the current week. However, if the user does not have a VE balance yet, their `_userTokenTimeCursor` will actually be set to the very first week the `token` was indexed at. Example: 1. Year 1: `STG` is distributed via `FeeDistributor`. 2. Year 10: Alice submits a transaction to stake her first `veSTG` tokens. 3. Exploiter Bob frontruns this transaction and calls `claimToken(alice, STG)`. 4. Alice's `_userTokenTimeCursor` is set to 10 years in the past, which requires her to slowly catch up week-by-week for hundreds of weeks.
Recommendation	Consider not permitting claiming if the user does not have checkpoints yet (e.g. their maximum epoch is still zero), or consider not permitting third-party claims by default. Consider carefully checking whether there are other cases where the token `startTime` and address would be erroneously returned. The most sensible method seems to be to simply revert on line 411 (`if (maxUserEpoch == 0) return`) instead of `return`, as this reduces the state-space. Consider preventing third-party griefing altogether by making third-party claiming opt-in.

Resolution	RESOLVED Claiming via third-parties can now be disabled as an opt-out mechanism. More importantly, `_checkpointUserBalance` now reverts if the user has not yet checkpointed on the VE token by reverting on line 411.

Issue #02	**Distribution to weeks where no one is supplying to the VE token is completely unclaimable**
Severity	LOW SEVERITY
Description	There presently appears to be no check within the distribution function (`_checkpointToken`) to ensure any week that is assigned fees actually has users to claim them, i.e., there is no logic which prevents the assigning tokens to a week where no one has any VE tokens except for the very first week in a specific case during deployment. Secondly, it appears as if a portion of the distributable tokens is voided if no token `timeCursor` update occurs for more than 20 weeks. In this case, `timeSinceLastCheckpoint` is over 20 weeks while only the first 20 weeks will receive a share of the tokens to distribute. The sum of `((nextWeek - lastTokenTime) / timeSinceLastCheckpoint)` will in this scenario sum to less than 100%. Note that this secondary issue does not present itself as long as the deposit functions are used since these functions pre-index the `timeCursor` of the token before granting the tokens.
Recommendation	Consider whether it is worth addressing this issue. If there are never going to be such weeks, we understand that not addressing this issue is the cleanest to stay close to Balancer's original codebase. If desired, a governance function can be added to take out the tokens for such weeks. A check on whether a week has a `_tokensPerWeek` value but no `_veSupplyCache` combined with a new mapping to ensure these values can only be claimed once appears sufficient.

More intrusively, a carry-over mechanism could be considered within `_checkpointToken` where empty weeks are not assigned tokens, but instead those tokens are assigned to a non-empty week after it. Logic needs to be added to ensure that these tokens are eventually assigned or, if not assigned, subtracted from `cachedBalance`.

For the secondary issue, it may make sense to grant these tokens to the final epoch of the 20 epochs, or to skip to the most recent week and grant everything there in this scenario. Alternatively and much easier, consider simply always using the `depositToken` functions which pre-index the `timeCursor` and seem to avoid this secondary issue altogether.

Resolution

RESOLVED

A simple governance function has been added to freely take out any tokens in the contract. It should be noted that this makes the contract more centralized as it permits any tokens within the contract to be taken out by the `owner`.

The team should be extremely vigilant with safeguarding this ideally multi-signature account. No safeguards were added that prevent the owner from taking out all tokens within this contract. The function also lacks an event.

Issue #03	Contract is incompatible with multi-address tokens like TUSD
Severity	INFORMATIONAL
Description	An uncommon edge-case within ERC20 implementations are tokens which have two interface addresses sharing a single storage. An example is TUSD: https://medium.com/chainsecurity/trueusd-compound-vulnerability-bc5b696d29e2 In case the team ever wishes to distribute such a token, this distribution will be exploitable to the point where VE holders can claim double the amount of tokens they are owed, thus breaking the distribution business logic.
Recommendation	Consider whether such tokens will ever possibly be added. In this case, an explicit token address whitelist makes sense. A decentralized alternative to a whitelist is possible through omitting the `balanceOf` logic within `_checkpointToken` and instead moving it to `depositToken` and `depositTokens` in a before-after methodology where only actual balance increments during these operations are recorded within the cache. Interestingly, if the removal of the `balanceOf` from the checkpoint logic is done carefully, it could be seen as an improvement by itself. This would for example allow for the distribution of the remaining portion of a token if its balance is decreased exogenously (eg. through a rebase) compared to simply fully bricking on the `.sub` statement.
Resolution	RESOLVED The client has stated they will not use such tokens. No changes were made.

Issue #04	The `uint` keyword should not be used
Severity	INFORMATIONAL
Description	Within the `FeeDistributor`, the `uint` keyword is used. It is highly recommended to only use the `uint256` keyword to make it clearer.
Recommendation	Replace all the use of the uint keyword to `uint256`.
Resolution	RESOLVED

Issue #05	Additional validation can reduce the state-space for the contract making it more theoretically secure
Severity	INFORMATIONAL
Description	The first epoch within the `VotingEscrow` contract is set to `block.timestamp`, which may be after the `startTime` of the `FeeDistributor`. This edge-case would allow for the state-space of the contract to be more expansive and potentially erroneous as the first epoch would be returned through things like the binary search for timestamps before the first epoch. The binary search is always supposed to return the epoch before the provided timestamp and ideally such states where no such epoch exists should simply not be possible. Consider therefore preventing this possibility. This issue is not provided in depth as it appears impossible given that `VeSTG` is already deployed. Furthermore, when deployed, there exists a period where no checkpoints have occurred yet of things like the total VE supply. Many of the aforementioned recommendations have no explicitly visible way of getting exploited. However, since there are no downsides to reducing the state-space, it might be valuable to implement them. It should finally be noted that many getters within the contract implicitly return 0 for inputs which are not yet valid (eg. timestamps in the future or timestamps yet to be indexed). It may make sense to add requirements though this is an opinionated design decision and either choice doesn't necessarily make the contract more secure.
Recommendation	Add `require(startTime > votingEscrow.point_history[0].ts)` to the constructor to prevent the aforementioned edge-case. Consider also immediately calling `_checkpointTotalSupply` to further reduce the state-space and eliminate the transient state where no total supply has yet been indexed.
Resolution	RESOLVED The requirement has been added. The checkpoint was however not added, but will be called manually after deployment by the team. This is because the immutable variable is not callable with their Solidity version.

Issue #06	Allowing anyone to claim on behalf for other users might be annoying
Severity	INFORMATIONAL

Location	Lines 246 and 263
	`function claimToken(address user, IERC20 token) external`
	`function claimTokens(address user, IERC20[] calldata tokens) external`

Description	The contract presently allows anyone to claim tokens for anyone. Certain users may not want to allow this as it may be considered a taxable event. Even if they do not mind, they may not want to spend the time to report all these small transactions if someone starts calling this function every single week for them, instead of at their desired interval. Furthermore, certain integrations (e.g. contracts building on top of the `FeeDistributor`) might accidentally not realize that these contracts are not the only contracts able to call this function, potentially opening them up to unnecessary vulnerabilities.
Recommendation	Consider whether it makes sense for third-party claiming to be opt-in or approval based.
Resolution	RESOLVED The client has chosen for an opt-out mechanism.

Issue #07	Gas optimizations
Severity	INFORMATIONAL

Description	Line 335
	`require(block.timestamp > _startTime, "Fee distribution has not started yet");`
	
	This check can be done a few lines earlier to save gas in the negative case.
	
	Line 535 (example)
	`IVotingEscrow votingEscrow = _votingEscrow;`
	
	`_votingEscrow` is cached unnecessarily in various functions as this variable is immutable.
	
	Line 627
	`return _roundDownTimestamp(timestamp + 1 weeks - 1);`
	
	The 1 weeks - 1 section can be made constant to save on gas.

Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	RESOLVED

Issue #08	Typographical issues
Severity	INFORMATIONAL

Description	The `UserState` struct contains two variables that can not be read through a public function: `lastEpochCheckpointed` and `startTime`.
	
	——
	
	The `TokenState` struct contains variables that can not be read through a public function: `startTime` and `cachedBalance`.
	
	——
	
	The `_startTime` variable cannot be read through a public function.
	
	——
	
	Line 23
	`* holders simply transfer the tokens to the`
	`` `FeeDistributor` contract and then call `checkpointToken`. ``
	
	The comments within the code recommend to not deposit through this avenue and instead use `depositToken`.
	
	Lines 251-252
	`uint amount = _claimToken(user, token);`
	`return amount;`
	
	This can be re-written into a single line by immediately returning the value. Depending on the compiler used, this can also save on gas.

Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY