



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero (AASTGClaim)

08 March 2023



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	4
1.3 Findings Summary	5
1.3.1 AASTGClaim	6
2 Findings	7
2.1 AASTGClaim	7
2.1.1 Privileged Functions	7
2.1.2 Issues & Recommendations	8



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 advocacy 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 LayerZero's AASTGClaim contract on the Ethereum 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	LayerZero
URL	https://layerzero.network/
Platform	Ethereum
Language	Solidity
Preliminary Contracts	https://github.com/ryanzarick/stargate/blob/1c730137f223460c114d44dbe52a5d3d60d6d081/contracts/AASTGClaim.sol
Final Contracts	https://github.com/ryanzarick/stargate/blob/f5ff4f3d035e4a2136db1af3a2e9fc5d605362a9/contracts/AASTGClaim.sol

1.2 Contracts Assessed

Name	Contract	Live Code Match
AASTGClaim	0x879ca077b05579fa68aab2827bed21f5f50eacbb	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	1	1	-	-
● Medium	0	-	-	-
● Low	2	-	-	2
● Informational	4	-	-	4
Total	7	1	-	6

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 AASTGClaim

ID	Severity	Summary	Status
01	HIGH	All contract functionality is presently broken due to a reentrancy guard gridlock reverting every single call	✓ RESOLVED
02	LOW	Governance risk: Contract owner can withdraw all Stargate tokens	ACKNOWLEDGED
03	LOW	Governance risk: Merkle tree might allocate more tokens than are in the claim contract	ACKNOWLEDGED
04	INFO	Using a single hash function for the leafs is an anti-pattern	ACKNOWLEDGED
05	INFO	Typographical errors	ACKNOWLEDGED
06	INFO	Gas optimization	ACKNOWLEDGED
07	INFO	Griefing: Redemptions can be triggered for others without their consent	ACKNOWLEDGED

2 Findings

2.1 AASTGClaim

AASTGClaim is a linear vesting contract which allows the vesting of STG (Stargate) tokens to eligible recipients.

Recipients are considered eligible if the on-deployment merkle tree includes their address and their vesting amount. Recipients can solely enable their vest via `redeemWithProof` if the merkle tree includes those details for them.

Once a recipient has enabled their vesting, they can periodically call `redeem` to redeem their vested portion of STG tokens.



Based on the current implementation of the contract, tokens vest linearly for 26 weeks after a configured `VEST_START_TIME` which is the same for all users, regardless of when they call `redeemWithProof`.



Paladin did not audit the composition of the merkle proof.



2.1.1 Privileged Functions

- `withdrawFees [owner]`
- `transferOwnership [owner]`
- `renounceOwnership [owner]`

2.1.2 Issues & Recommendations

Issue #01	All contract functionality is presently broken due to a reentrancy guard gridlock reverting every single call
Severity	 HIGH SEVERITY
Description	<p>Both the <code>redeemWithProof</code> and <code>redeem</code> functions have a <code>nonReentrant</code> modifier. This modifier will revert any call to another function with the modifier as long as its function is not exited from. This is beneficial as it mitigates any risk for reentrancy exploits, apart from the code already being written in checks-effects-interactions (which sufficiently mitigates it as well).</p> <p>The <code>redeemWithProof</code> implementation calls <code>redeem</code> at the end.</p> <p><u>Line 45</u></p> <pre>redeem(_user);</pre> <p>However the reentrancy guard has already been primed at this point, which causes the <code>redeem</code> call to always revert due to an accidental reentrancy. All contract functionality is therefore broken within the version we received.</p>
Recommendation	Consider creating an internal <code>_redeem</code> function which is not guarded. Both external functions should call the <code>_redeem</code> function instead.
Resolution	 RESOLVED
	<p>The client has decided to remove the reentrancy guard from <code>redeemWithProof</code> given that the code already adheres to checks-effects-interactions.</p>

Issue #02 Governance risk: Contract owner can withdraw all Stargate tokens	
Severity	 LOW SEVERITY
Description	<p>The AASTGClaim contract will hold significant amounts of Stargate tokens. However, it contains an emergency feature where the contract owner can withdraw all these tokens again.</p> <p>We understand the need for such a feature, but it does pose a risk to holders of Stargate tokens or recipients of this contract. Therefore, included this issue to remind the team to secure the owner role with utmost care.</p>
Recommendation	Consider transferring ownership of the contract to a reputable multi-signature contract as soon as a significant amount of tokens has entered the contract.
Resolution	 ACKNOWLEDGED

Issue #03 Governance risk: Merkle tree might allocate more tokens than are in the claim contract	
Severity	 LOW SEVERITY
Description	There is no mechanism to ensure that the claim contract balance is sufficient to pay out everyone's claims. If the merkle tree is badly constructed or too little tokens are put in the contract, this would cause issues as users would not be able to redeem their tokens at some point.
Recommendation	Consider ensuring the contract has sufficient balance whenever redeemWithProof is called. An allocation counter would need to be added to keep track of the currently allocated contract balance, which ensures that anyone who has started their redemption will be able to fully complete it, unless the tokens are pulled.
Resolution	 ACKNOWLEDGED <p>The client indicated they will be mindful of this issue when setting this up.</p>


Issue #04**Using a single hash function for the leafs is an anti-pattern****Severity** INFORMATIONAL**Description**

The merkle tree leafs are encoded as keccak256(address user, uint256 amount). This is simple and secure but is considered an anti-pattern as it promotes the possibility of second preimage attacks.

We should note that such attacks are not possible within this codebase, but we still recommend adhering to best practices as it prevents the possibility of forgers shooting themselves in the foot when they adjust the leaf data. OpenZeppelin is also pushing double-hashing as a standard as well.

Recommendation

Consider using double-hashing or two different hash functions. The latter is historically recommended (e.g. Tornado's merkle tree) but we find it an equal anti-pattern as it increases the cryptographic attack surface.

Resolution ACKNOWLEDGED

The team has indicated they are fine with the current behavior as it is non-exploitable.



Severity

 INFORMATIONAL

Description

We have consolidated the typographical issues into a single issue to keep the report brief and readable.

Line 24

```
event Redeemed(address _sender, uint _stgAmount);
```

The sender address can be indexed for easier lookups.

Line 26

```
constructor(address _stgToken, bytes32 _merkleRoot) {
```

_stgToken can be provided as IERC20 to avoid casting it later on.

Line 36

```
// needs to be called first time a user redeems, sets the  
balance[_user] so subsequent vests dont require providing  
proof
```

"dont" should be "don't".

Line 68

```
function redeemable(address _user) external view returns  
(uint256) {
```


This whole function is repetitive in reference to a previous code section. Consider simply using redeemable in the redeem function to follow the DRY (don't repeat yourself) principle.

The contract uses both uint256 and uint interchangeably. It would be cleaner from a readability/code quality perspective to stick to a single one (we personally prefer to stick to uint256).

Recommendation

Consider fixing the typographical errors.

Resolution

 ACKNOWLEDGED

Issue #06	Gas optimization
Severity	INFORMATIONAL
Location	<u>Line 86</u> <pre>function _verify(bytes32 _leaf, bytes32[] memory _proof) internal view returns (bool) {</pre>
Description	_proof can be marked as calldata to save gas.
Recommendation	Consider implementing the gas optimization mentioned above.
Resolution	ACKNOWLEDGED

Issue #07	Griefing: Redemptions can be triggered for others without their consent
Severity	INFORMATIONAL
Description	<p>Redemptions can be triggered for anyone without their consent. This might be undesirable as certain recipients might want to only redeem at certain intervals for accounting purposes.</p> <p>More importantly, if a smart contract is ever set as a recipient (eg. a protocol which receives their tokens with a smart contract), this protocol might not account for the fact that redemptions can be triggered through other means than the protocol itself triggering them. This has historically led to exploits in more complicated codebases that are inter-dependent.</p>
Recommendation	Consider not allowing redemptions for other users if this behavior is not desired.
Resolution	ACKNOWLEDGED



PALADIN
BLOCKCHAIN SECURITY