etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Aerodrome (AERO)
Website: aerodrome.finance
Platform: Base Chain Network
Language: Solidity
Date: June 5th, 2024

Table of contents

Introduction 4
Project Background 4
Audit Scope 5
Code Audit History 6
Severity Definitions 6
Claimed Smart Contract Features 7
Audit Summary 8
Technical Quick Stats 9
Business Risk Analysis 10
Code Quality 11
Documentation 11
Use of Dependencies 11
AS-IS overview 12
Audit Findings 13
Conclusion 16
Our Methodology 17
Disclaimers 19
Appendix

- Code Flow Diagram 20
- Slither Results Log 21
- Solidity static analysis 22
- Solhint Linter 23

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Aerodrome smart contract from aerodrome.finance was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on June 5th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

Website Details

Aerodrome Finance is a decentralized finance (DeFi) platform focused on providing various financial services on the blockchain. It features tools for trading, liquidity provision, and yield farming. Users can engage with the platform to manage digital assets, earn rewards, and participate in a decentralized financial ecosystem.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Code Details

- The `AeroToken` contract inherits from `ERC20` and implements the `IAero` interface.
- The `minter` address is set at deployment and can mint new tokens.
- The `mint` function allows the minter to mint new tokens to a specified address.
- The `_beforeTokenTransfer` and `_afterTokenTransfer` functions can be overridden for any custom logic you may want to add before or after token transfers.

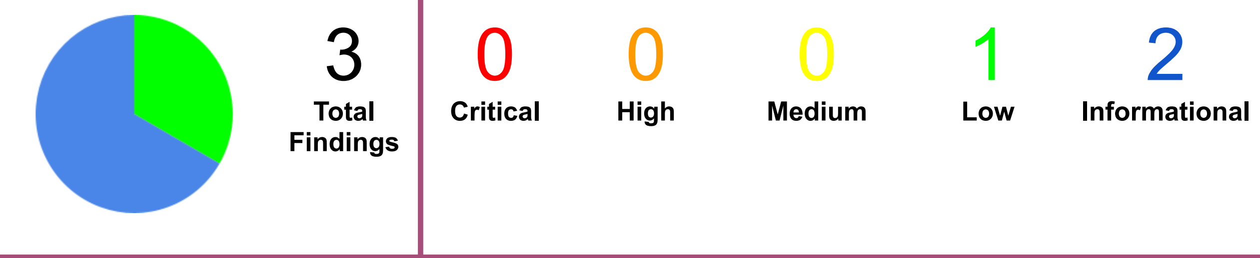
This contract makes use of OpenZeppelin's `ERC20` implementation and utilities to ensure security and standard compliance. Make sure you have the required OpenZeppelin contracts in your project.

Audit scope

Name	**Code Review and Security Analysis Report for Aerodrome (AERO) Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	Aero.sol
Smart Contract Code	0x940181a94a35a4569e4529a3cdfb74e38fd98631
Audit Date	June 5th,2024
Audit Result	**Passed**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Code Audit History



Severity Definitions

0	**Critical**		Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0	**High**		High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0	**Medium**		Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
1	**Low**		Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
2	**Lowest / Informational / Best Practice**		Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Aerodrome • Symbol: AERO • Decimals: 18	**YES, This is valid.**
Ownership Control: • Set the minter address. • Mint unlimited token.	**YES, This is valid.**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured"**.Also, these contracts contain owner control, which does not make them fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 1 low, and 2 very low-level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Moderated
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	No
Fee Check	Not Detected
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	No
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	Yes
Is it a Proxy Contract?	No
Is it used Open Source?	No
External Call Risk?	No
Balance Modifiable?	No
Can Take Ownership?	No
Ownership Renounce?	No
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: PASSED

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Aerodrome are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Aerodrome.

The EtherAuthority team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given an Aerodrome smart contract code in the form of a basescan web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

AS-IS overview

Aero.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	setMinter	external	Emit an appropriate events	Refer Audit Findings
3	mint	external	Mint unlimited token	Refer Audit Findings
4	permit	write	Passed	No Issue
5	nonces	read	Passed	No Issue
6	DOMAIN_SEPARATOR	external	Passed	No Issue
7	_useNonce	internal	Passed	No Issue

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

[L-01] Mint unlimited token:

```
function mint(address account, uint256 amount) external returns
(bool) {
        if (msg.sender != minter) revert NotMinter();
        _mint(account, amount);
        return true;
    }
```

Description:
There is no limit for minting tokens. Thus the owner can mint unlimited tokens to any account.

Recommendation: There should be a limit for minting or need to confirm, if it is a part of the plan then disregard this issue.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Very Low / Informational / Best practices:

[I-01] Emit appropriate events:

```
  /// @dev No checks as its meant to be once off to set minting
rights to BaseV1 Minter
    function setMinter(address _minter) external {
       if (msg.sender != minter) revert NotMinter();
        minter = _minter;
    }
```

Description:

Ensure that state-changing functions emit appropriate events.

Recommendation: We suggest adding the event in the setMinter function.

[I-02] NatSpec comments:

Description:

Add NatSpec comments for all public and external functions for clarity.

Recommendation: We suggest first checking all public and external functions. Are they commented properly?

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble.

Centralized Decentralized

You are here

The following are owner functions:

Aero.sol

- setMinter: Updated minter address only by the owner.
- mint: The owner can mint tokens.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Conclusion

We were given a contract code in the form of a basescan web link. And we have used all possible tests based on given objects as files. We observed 1 low and 2 Informational issues in the smart contracts. but those are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

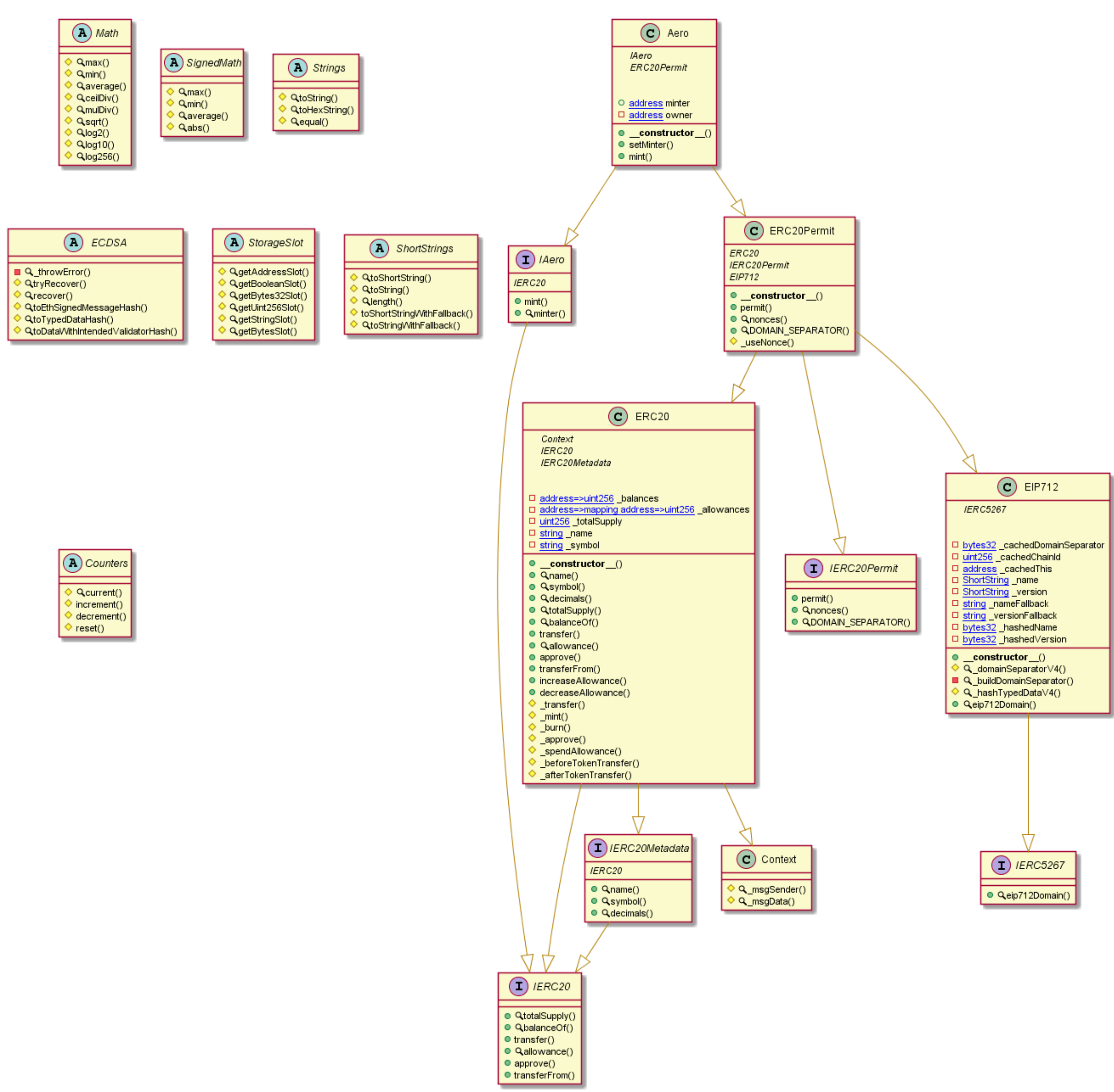
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Aerodrome



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Aero.sol

```
INFO:Detectors:
ERC20Permit.constructor(string).name (Aero.sol#1689) shadows:
	- ERC20.name() (Aero.sol#197-199) (function)
	- IERC20Metadata.name() (Aero.sol#110) (function)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Aero.setMinter(address)._minter (Aero.sol#1756) lacks a zero-check on :
		- minter = _minter (Aero.sol#1758)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Pragma version0.8.19 (Aero.sol#6) necessitates a version too recent to be trusted. Consider
deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function IERC20Permit.DOMAIN_SEPARATOR() (Aero.sol#559) is not in mixedCase
Function ERC20Permit.DOMAIN_SEPARATOR() (Aero.sol#1726-1728) is not in mixedCase
Variable ERC20Permit._PERMIT_TYPEHASH_DEPRECATED_SLOT (Aero.sol#1682) is not in
mixedCase
Parameter Aero.setMinter(address)._minter (Aero.sol#1756) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
Aero.owner (Aero.sol#1748) should be immutable
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-decl
ared-immutable
INFO:Slither:Aero.sol analyzed (17 contracts with 93 detectors), 75 result(s) found
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

Aero.sol

Inline assembly:
The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
Pos: 1419:15:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 1703:23:

Gas costs:
Gas requirement of function Aero.permit is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1694:11:

Gas costs:
Gas requirement of function Aero.mint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1761:11:

Similar variable names:
Aero.mint(address,uint256) : Variables have very similar names "account" and "amount".
Pos: 1763:30:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 1703:15:

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

Aero.sol

```
Compiler version 0.8.19 does not satisfy the ^0.5.8 semver requirement
Pos: 1:5
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:188
Error message for require is too long
Pos: 9:334
Error message for require is too long
Pos: 9:357
Error message for require is too long
Pos: 9:358
Error message for require is too long
Pos: 9:363
Error message for require is too long
Pos: 9:412
Error message for require is too long
Pos: 9:417
Error message for require is too long
Pos: 9:443
Error message for require is too long
Pos: 9:444
Code contains empty blocks
Pos: 94:482
Code contains empty blocks
Pos: 93:498
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 13:623
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 13:643
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 13:657
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 13:956
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 17:962
Error message for revert is too long
Pos: 13:1041
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

```
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 13:1073
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1179
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1209
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1291
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1301
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1311
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1321
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1331
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1341
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1351
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1361
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 9:1418
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1540
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1688
Code contains empty blocks
Pos: 55:1688
Avoid making time-based decisions in your business logic
Pos: 17:1702
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:1749
```

Software analysis result:

This software reported many false positive results and some were informational issues. So, those issues can be safe

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io