QuillAudits

Audit Report
April, 2023

For

AUMENT

Table of Content

Executive Summary 01

Checked Vulnerabilities 03

Techniques and Methods 04

Manual Testing 05

A. Contract - Administrable.sol 05

B. Contract - AumentWalletContract.sol 06

C. Contract - ERC20Aument.sol 09

D. Contract - EscrowContract.sol 10

E. Contract - EscrowFactoryProxy.sol 12

F. Contract - EscrowProxy.sol 13

G. Contract - ProxyStorage.sol 14

H. Contract - RevertMsg.sol 15

I. Common Issues and Recommendations 15

Functional Tests 18

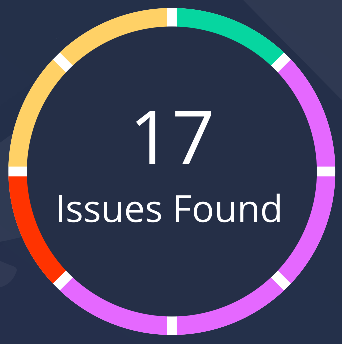
Automated Tests 19

Closing Summary 21

About QuillAudits 22

Executive Summary

Project Name	Aument
Overview	The Aument codebase includes a token and loan facilitation agreement (loan is provided by the bank) featuring a primary function of assisting token holders in obtaining loans using their tokens as collateral. These tokens are secured in a collateral (escrow) contract until the loan term reaches its conclusion. At this point, the contract transitions to an inactive state (rendered non-functional) and the tokens are returned to the lender or borrower, contingent upon the fulfillment of specific conditions.
Timeline	November 22, 2022 - April 14, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze Aument codebase for quality, security, correctness, and exchange listing. *https://gitlab.com/aument/crypto-contracts/-/tree/master/contracts* "master" branch with commit hash 03573264991c49eb4608fdca9cef7ded9fa44cf9
Fixed In	*https://gitlab.com/aument/crypto-contracts/-/tree/audit-feedback*



🟥 High 🟨 Medium 🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	1	1	1	3
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	4	2	4

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open
Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved
These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged
Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved
Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - Administrable.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

1. No need for equality in comparison

Description

The require statement expects a true value else it reverts with an error message. In this case, if isAdmin(msg.sender) returns true, the modifier would allow the rest of the code to run to completion. As such, there is no need to equate the return value of isAdmin() 'true' to true.

Remediation

Having the code in this format will provide some gas savings.

```
modifier onlyAdmin() {
      require(isAdmin(msg.sender), _errorMessage);
      _;
}
```

Status

Acknowledged

2. Lack of comments

Description

The codebase could provide more detail with descriptions and comments. It is generally good practice to have inline comments to make code easily readable.

Remediation

Consider including more comments in the codebase.

Status

Resolved

B. Contract - AumentWalletContract.sol

High Severity Issues

3. Reentrancy risk

Description

The withdrawToken( ) and withdrawEther( ) functions are less likely to pose a threat but proxyCallWithValue( ) and proxyCallWithoutValue( ) make external calls that can pose potential reentrancy risks.

Remediation

Consider using a reentrancy guard or following the Check-Effect-Interact pattern to mitigate possible loss of funds.

Status

Resolved

Medium Severity Issues

4. Missing input validation checks

Description

The constructor, proxyCallWithValue, proxyCallWithoutValue functions do not have any input validation mechanisms. Zero addresses can be passed in without reverting.

Remediation

Before critical changes to the contract state, certain checks can be made; for example: to confirm the previous value is not the same as the new one to be added, zero address checks, integer over or underflows.

Status

Resolved

Low Severity Issues

5. Interface Naming

Description

The IERC20Aument interface is defined inline here but is also a standalone contract with an ABI.

Remediation

Consider renaming the interface or importing the same functions in both.

Status

Resolved

Informational Issues

6. Gas Optimization

Description

The code within the if block only runs if the check passes.

Remediation

If “success != true” can be rewritten as “!success” for minor gas savings as well.

Status

Resolved

7. Lack of comments

Description

The codebase could provide more detail with descriptions and comments. It is generally good practice to have inline comments to make code easily readable.

Remediation

Consider including more comments in the codebase.

Status

Acknowledged

C. Contract - ERC20Aument.sol

High Severity Issues

8. Centralization

Description

The contract is heavily dependent on admin accounts, and if any of these admins are an EOA whose private key is compromised, the entire platform is at risk of being exploited. Some of these risks are: arbitrary token mints till the cap is reached, AumentWalletAddress being reset at will, transferFee adjustments, removing other whitelisted addresses.

Remediation

We advise the client to carefully manage the admin account private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets

Status

Acknowledged

Medium Severity Issues

9. Missing input validation checks in setAumentWalletAddress( )

Description

The critical function setAumentWalletAddress( ) lacks input validation as well. There are no checks for what address can be passed, which could even be a zero address. Since the transfer( ) and transferFrom( ) functions do not charge fees when called with this address, a malicious admin can pass in their own address and circumvent paying the fees for the transaction.

Remediation

Include address validity checks, this can be implemented via require statements.

Status

Resolved

Low Severity Issues

10. Math calculation misstep

Description

Fees for the Aument token are intended to never go above 5%. This gives a range from 0 %- 5% with 5% inclusive. The current require check would fail if 5% (50000) is passed in.

Remediation

Update the require check to be less or equal to (<=) and not just less than (<) 5% as it currently is.

Status

Resolved

Informational Issues

11. Emit events for notable changes to state

Description

When state is changed, it is advisable to keep logs to make reference to.

Remediation

Events can be emitted to keep track of these changes.

Status

Acknowledged

D. Contract - EscrowContract.sol

High Severity Issues

No issues found

Medium Severity Issues

12. Contract creation dependence

Description

EscrowContract.sol depends on the variables _escrowFactoryAddress and _erc20AumentContractAddress to be initialized before it gets deployed. This is done through the Escrow Proxy contract constructor.

Remediation

Ensure the EscrowProxy.sol contract gets deployed before EscrowContract.sol is deployed. An even better alternative would be to alter the contract architecture to ensure this dependence does not exist.

Status

Acknowledged

Low Severity Issues

No issues found

Informational Issues

13. Modifier only used once

Description

It is common practice to use modifiers to reduce repetitive code that cuts across multiple functions in a contract/codebase. The onlyFactory() modifier is only called once through the scope of the entire codebase.

Remediation

Consider including the modifier code in the single function it is used in.

Status

Resolved

E. Contract - EscrowFactoryProxy.sol

High Severity Issues

No issues found

Low Severity Issues

14. Assembly Usage

Description

The EscrowProxy.sol contract contains a block of assembly code which bypasses all checks of the solidity compiler and is not recommended for use. The assembly code attempts to create new contracts using the CREATE2 opcode. With this,there is a likelihood of deploying to a previously existing address with a wiped state due to the feature/bug introduced after the Constantionple upgrade. *Reference*.

Remediation

Consider alternatives to the assembly block implemented here, or implement techniques to mitigate the risk described above.

Status

Acknowledged

Informational Issues

15. Gas Optimization

Description

The code within the if block only runs if the check passes.

Remediation

If “success != true” can be rewritten as “!success” for minor gas savings as well.

Status

Resolved

F. Contract - EscrowProxy.sol

High Severity Issues

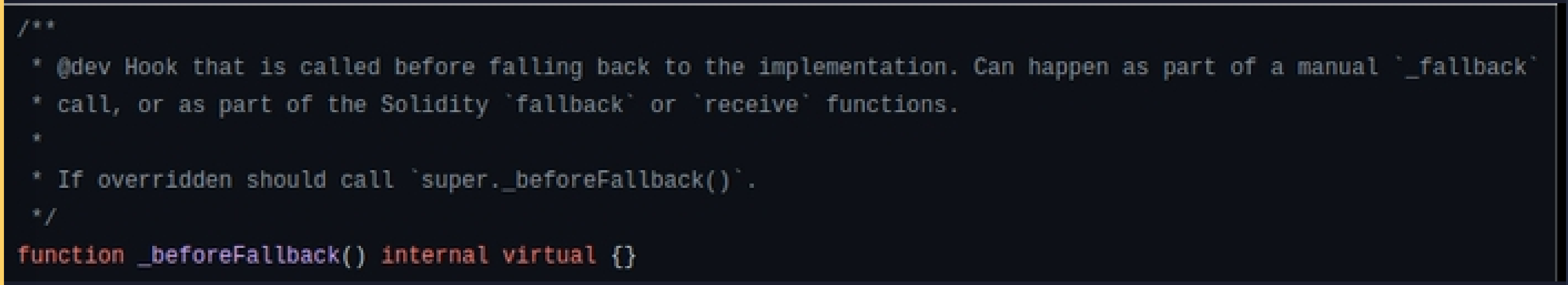
No issues found

Medium Severity Issues

16. Inaccurate code behaviour

Description

The OpenZeppelin Proxy contract details the need to call super._beforeFallback() when the _beforeFallback() function is overridden in any inheriting contracts. The EscrowProxy.sol contract overrides _beforeFallback on line 18 but does not call super._beforeFallback(). *Reference.*



```
/**
 * @dev Hook that is called before falling back to the implementation. Can happen as part of a manual `_fallback`
 * call, or as part of the Solidity `fallback` or `receive` functions.
 *
 * If overridden should call `super._beforeFallback()`.
 */
function _beforeFallback() internal virtual {}
```

Remediation

Consider updating the code to match up with recommendations made by external contracts and dependencies for optimal functionality.

Status

Resolved

17. Missing input validation checks

Description

The constructor does not have any input validation mechanisms. Zero addresses can be passed in without reverting.

Remediation

Consider implementing address checks, possible using require statements.

Status

Resolved

Low Severity Issues

No issues found

Informational Issues

No issues found

G. Contract - ProxyStorage.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

H. Contract - RevertMsg.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

I. Common Issues and Recommendations

18. Unlocked pragma (pragma solidity ^0.6.6)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all of the in-scope contracts have an unlocked pragma, it is recommended to lock all.

19. Loan requests

Description

The current contract logic does not cover for loan requests made as it is done offchain. The admins can approve or deny loans at will.

Remediation

Consider a means of decentralizing the flow of approvals and rejections of loans, and devise means to make them transparent/available.

Client Comment - The process of applying for a loan begins with the client contacting his bank for a loan, which, after the preliminary confirmation for issuing the loan, provides the client with information about the collateral related to that loan. Then the client addresses us regarding collateral coverage. Then, we sign a contract with the client that includes all the details related to collateral coverage, and subsequently a reference number is created under which the entire procedure is conducted, still all off-chain. After that, Aument Ag contacts the client's bank regarding depositing gold which will cover the collateral.
Only at the moment when the client sends Aument tokens related to the collateral is the moment when the system creates a special and unique escrow wallet where the tokens will be deposited during the entire duration of the loan.
In case the client does not send Aument tokens, the system is not activated and therefore there is no need to mark or monitor the procedure since it has not even started.
The system only has three positions that indicate completed, default and ongoing loan. So, the rejected position does not exist, considering that the initial part of the procedure is done off-chain and the system does not enter and does not monitor or trace data about such a procedure, which has never been activated.

20. Token prices and automatic price updates

Description

Token prices are based on the mechanics shown in their whitepaper. It is more advisable to have token prices obtained by means of decentralized price oracles (e.g. Chainlink) where possible. Also, for cases where token prices could be altered due to supply, a Rebase token mechanism can be implemented.

Reference

Rebase | Alexandria, What Is a Rebase/Elastic Token?

Client Comment - We want to emphasise that the Aument price is primarily related to the dynamics of demand and offer of Aument tokens on the exchange platform, but that also includes the possibility of automatic price correction based on the gold price throughout an algorithm which communicates with METALS API in order to maintain a solid collateral supporting the value of the token at all times. Our business model is based of having physical gold as the asset that supports the value of Aument Token. Metals-API is the #1 resource for real-time precious metals rates.
The value of Aument tokens can rise or fall depending on the market speculation within maximum 10%.
In this way, we enable our clients to benefit from the fluctuation of the gold price on the market which is statistically rising over time, and protect the clients over speculations that could decrease the value of the token.
The Aument token is backed by 90% of real gold deposited in the bank by Aument AG as a guarantee.
The whole dynamic of price correction is performed off-chain and is not included in the smart contract, that acts primarily as a protection by maintaining the connection with the collateral asset value of gold in real time.

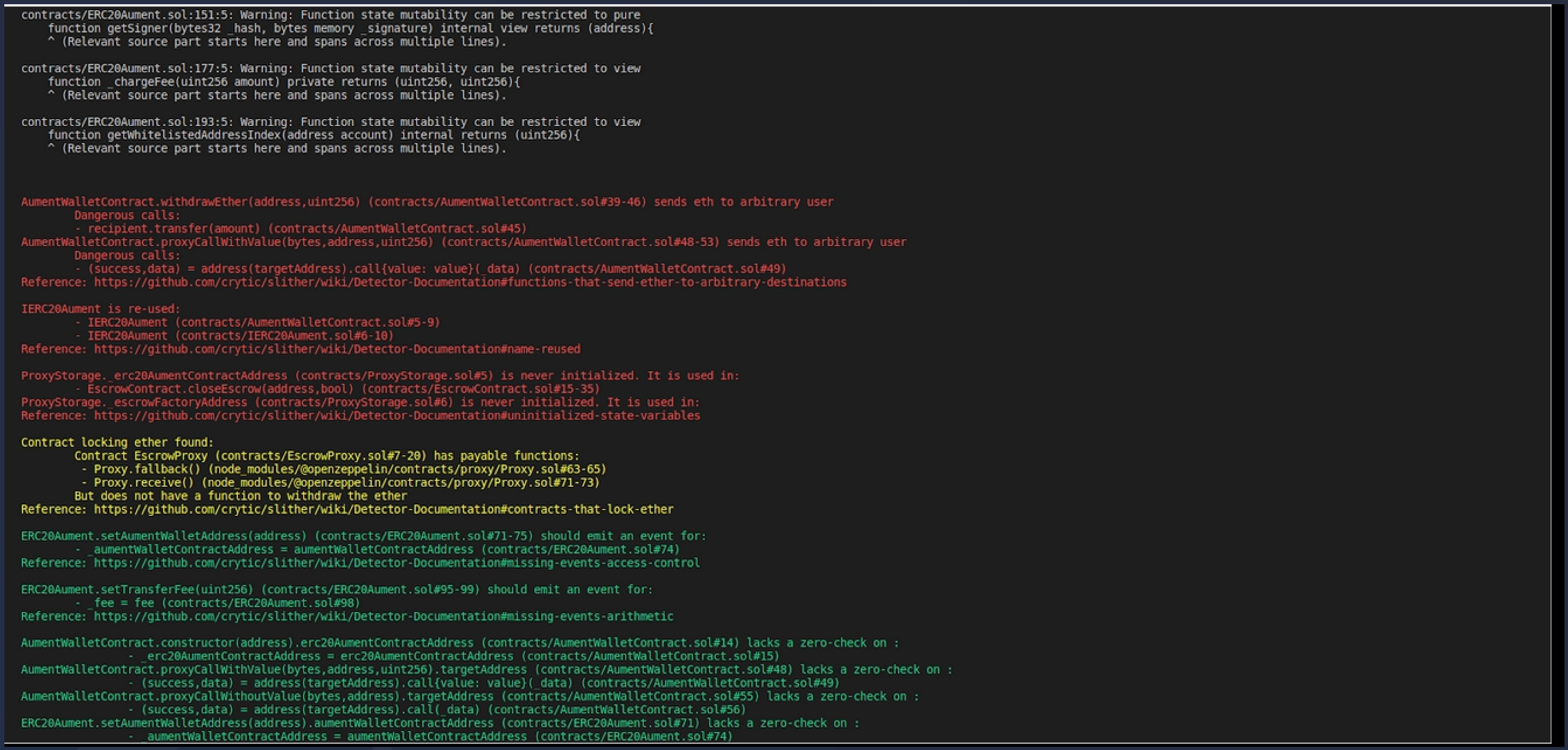
Functional Testing

Some of the tests performed are mentioned below:

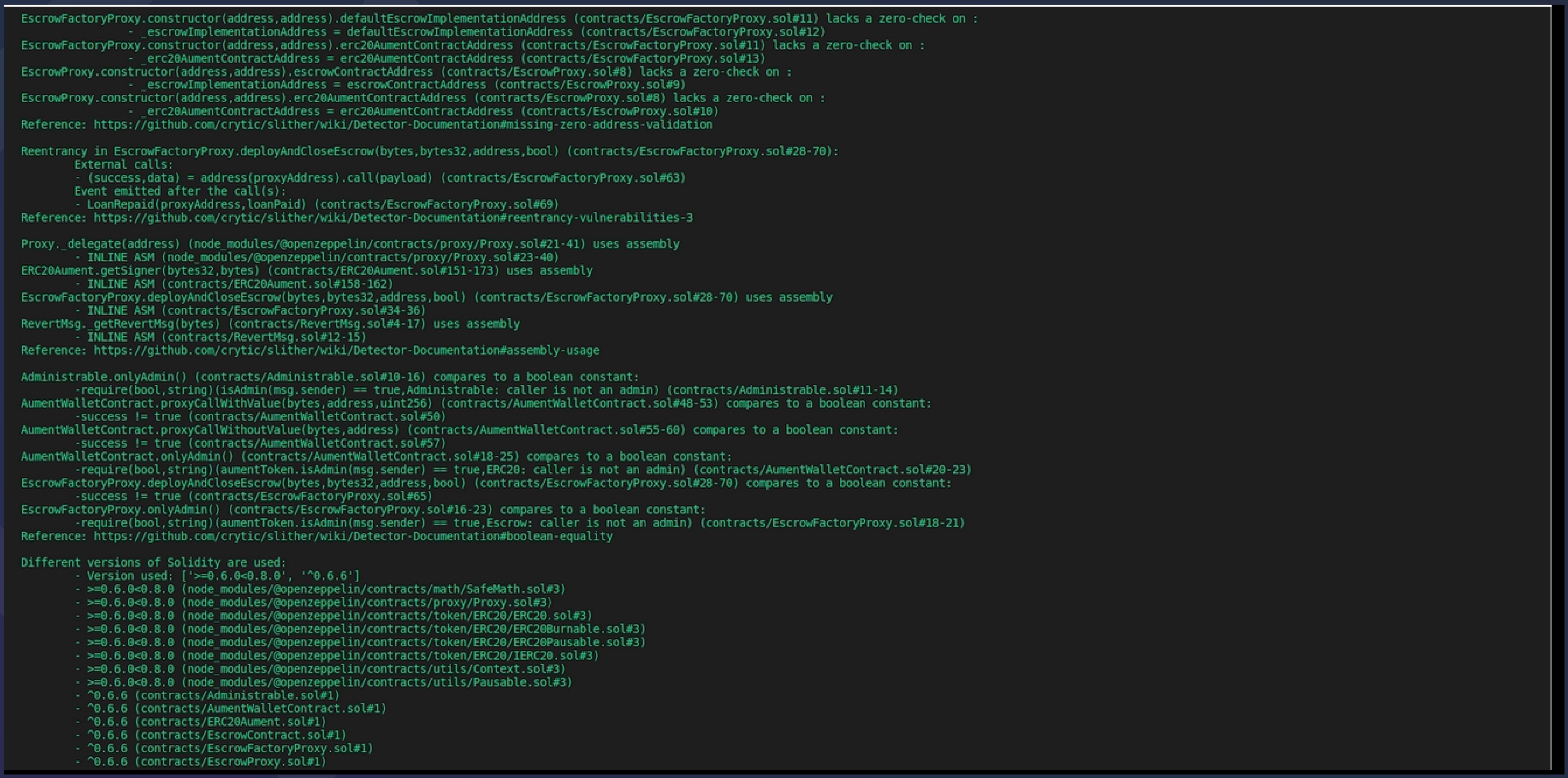
- ✓ should not restore admin privileges by caching old admins
- ✗ should not initialize with zero addresses
- ✓ should not exceed total supply
- ✓ should not mint new tokens when cap is reached
- ✓ should restrict functions to non-admins when paused
- ✓ should ensure ecrecover does not return dead address
- ✓ should restrict functions to non-admins when paused

Automated Tests

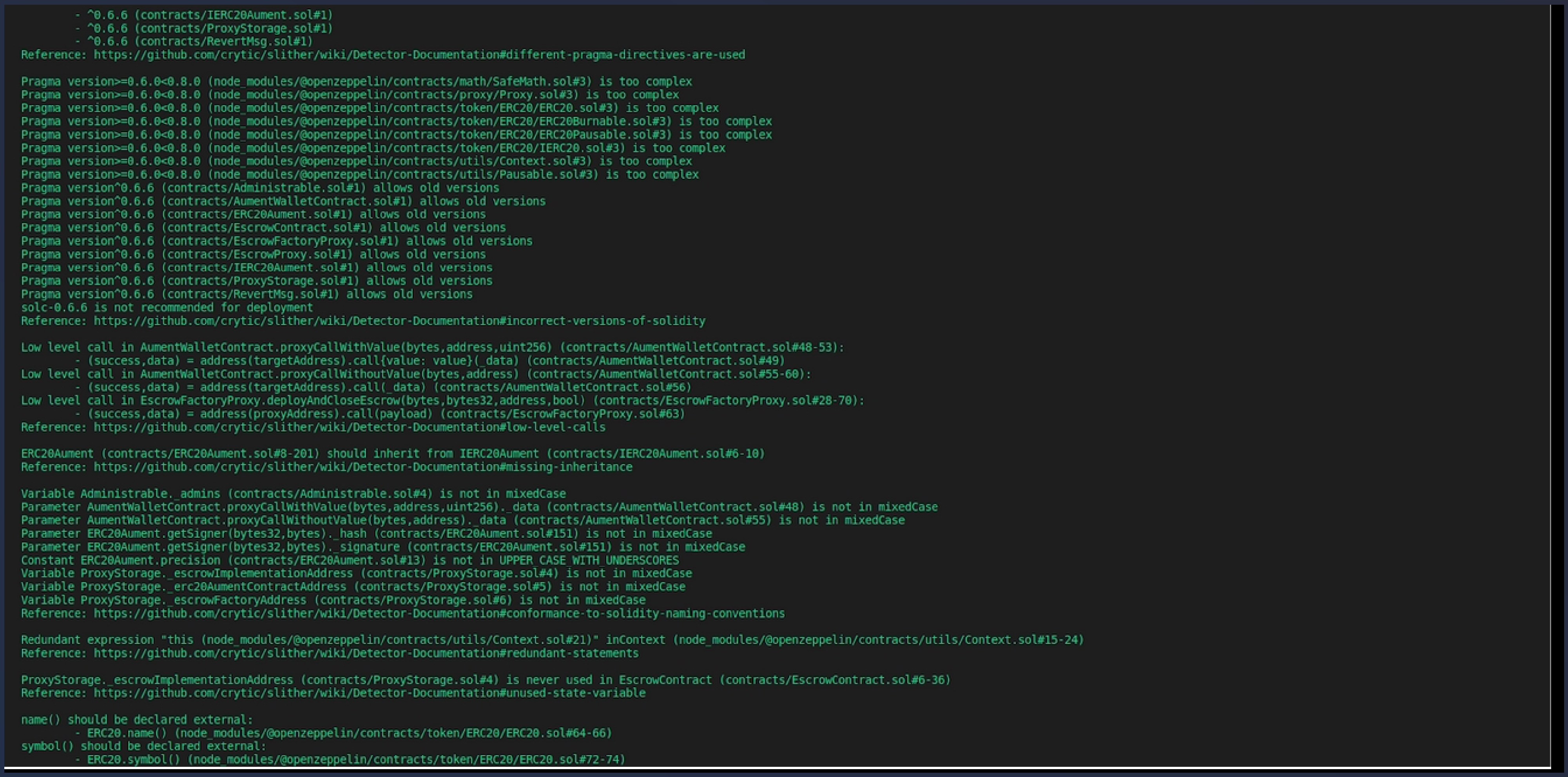
No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.



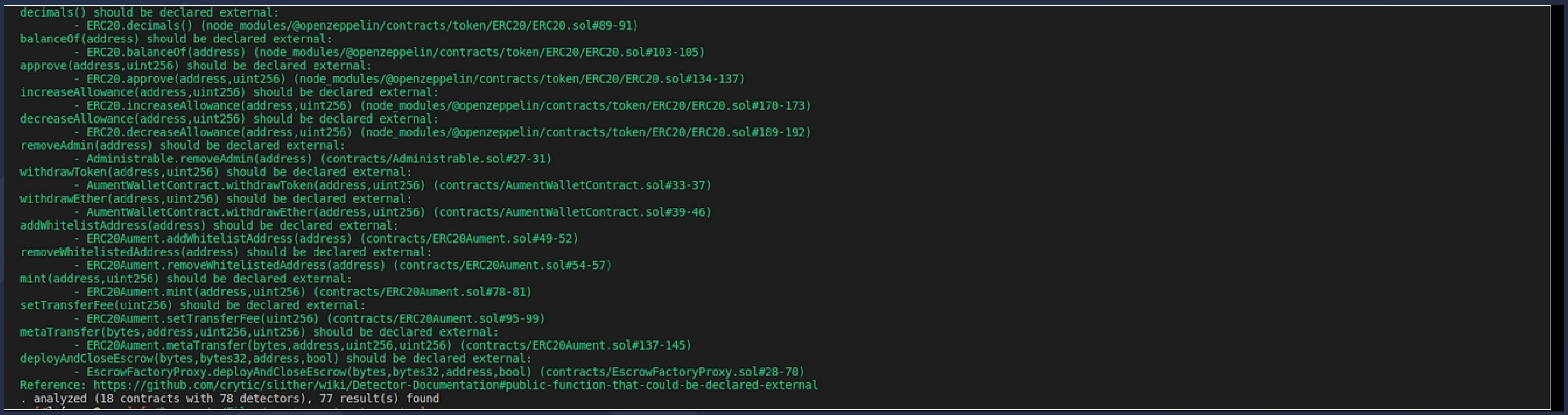
```
contracts/ERC20Aument.sol:151:5: Warning: Function state mutability can be restricted to pure
    function getSigner(bytes32 _hash, bytes memory _signature) internal view returns (address){
    ^ (Relevant source part starts here and spans across multiple lines).

contracts/ERC20Aument.sol:177:5: Warning: Function state mutability can be restricted to view
    function _chargeFee(uint256 amount) private returns (uint256, uint256){
    ^ (Relevant source part starts here and spans across multiple lines).

contracts/ERC20Aument.sol:193:5: Warning: Function state mutability can be restricted to view
    function getWhitelistedAddressIndex(address account) internal returns (uint256){
    ^ (Relevant source part starts here and spans across multiple lines).



AumentWalletContract.withdrawEther(address,uint256) (contracts/AumentWalletContract.sol#39-46) sends eth to arbitrary user
        Dangerous calls:
        - recipient.transfer(amount) (contracts/AumentWalletContract.sol#45)
AumentWalletContract.proxyCallWithValue(bytes,address,uint256) (contracts/AumentWalletContract.sol#48-53) sends eth to arbitrary user
        Dangerous calls:
        - (success,data) = address(targetAddress).call{value: value}(_data) (contracts/AumentWalletContract.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

IERC20Aument is re-used:
        - IERC20Aument (contracts/AumentWalletContract.sol#5-9)
        - IERC20Aument (contracts/IERC20Aument.sol#6-10)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused

ProxyStorage._erc20AumentContractAddress (contracts/ProxyStorage.sol#5) is never initialized. It is used in:
        - EscrowContract.closeEscrow(address,bool) (contracts/EscrowContract.sol#15-35)
ProxyStorage._escrowFactoryAddress (contracts/ProxyStorage.sol#6) is never initialized. It is used in:
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

Contract locking ether found:
        Contract EscrowProxy (contracts/EscrowProxy.sol#7-20) has payable functions:
         - Proxy.fallback() (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#63-65)
         - Proxy.receive() (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#71-73)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

ERC20Aument.setAumentWalletAddress(address) (contracts/ERC20Aument.sol#71-75) should emit an event for:
        - _aumentWalletContractAddress = aumentWalletContractAddress (contracts/ERC20Aument.sol#74)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

ERC20Aument.setTransferFee(uint256) (contracts/ERC20Aument.sol#95-99) should emit an event for:
        - _fee = fee (contracts/ERC20Aument.sol#98)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

AumentWalletContract.constructor(address).erc20AumentContractAddress (contracts/AumentWalletContract.sol#14) lacks a zero-check on :
                - _erc20AumentContractAddress = erc20AumentContractAddress (contracts/AumentWalletContract.sol#15)
AumentWalletContract.proxyCallWithValue(bytes,address,uint256).targetAddress (contracts/AumentWalletContract.sol#48) lacks a zero-check on :
                - (success,data) = address(targetAddress).call{value: value}(_data) (contracts/AumentWalletContract.sol#49)
AumentWalletContract.proxyCallWithoutValue(bytes,address).targetAddress (contracts/AumentWalletContract.sol#55) lacks a zero-check on :
                - (success,data) = address(targetAddress).call(_data) (contracts/AumentWalletContract.sol#56)
ERC20Aument.setAumentWalletAddress(address).aumentWalletContractAddress (contracts/ERC20Aument.sol#71) lacks a zero-check on :
                - _aumentWalletContractAddress = aumentWalletContractAddress (contracts/ERC20Aument.sol#74)
```



```
EscrowFactoryProxy.constructor(address,address).defaultEscrowImplementationAddress (contracts/EscrowFactoryProxy.sol#11) lacks a zero-check on :
                - _escrowImplementationAddress = defaultEscrowImplementationAddress (contracts/EscrowFactoryProxy.sol#12)
EscrowFactoryProxy.constructor(address,address).erc20AumentContractAddress (contracts/EscrowFactoryProxy.sol#11) lacks a zero-check on :
                - _erc20AumentContractAddress = erc20AumentContractAddress (contracts/EscrowFactoryProxy.sol#13)
EscrowProxy.constructor(address,address).escrowContractAddress (contracts/EscrowProxy.sol#8) lacks a zero-check on :
                - _escrowImplementationAddress = escrowContractAddress (contracts/EscrowProxy.sol#9)
EscrowProxy.constructor(address,address).erc20AumentContractAddress (contracts/EscrowProxy.sol#8) lacks a zero-check on :
                - _erc20AumentContractAddress = erc20AumentContractAddress (contracts/EscrowProxy.sol#10)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in EscrowFactoryProxy.deployAndCloseEscrow(bytes,bytes32,address,bool) (contracts/EscrowFactoryProxy.sol#28-70):
        External calls:
        - (success,data) = address(proxyAddress).call(payload) (contracts/EscrowFactoryProxy.sol#63)
        Event emitted after the call(s):
        - LoanRepaid(proxyAddress,loanPaid) (contracts/EscrowFactoryProxy.sol#69)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Proxy._delegate(address) (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#21-41) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#23-40)
ERC20Aument.getSigner(bytes32,bytes) (contracts/ERC20Aument.sol#151-173) uses assembly
        - INLINE ASM (contracts/ERC20Aument.sol#158-162)
EscrowFactoryProxy.deployAndCloseEscrow(bytes,bytes32,address,bool) (contracts/EscrowFactoryProxy.sol#28-70) uses assembly
        - INLINE ASM (contracts/EscrowFactoryProxy.sol#34-36)
RevertMsg._getRevertMsg(bytes) (contracts/RevertMsg.sol#4-17) uses assembly
        - INLINE ASM (contracts/RevertMsg.sol#12-15)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Administrable.onlyAdmin() (contracts/Administrable.sol#10-16) compares to a boolean constant:
        -require(bool,string)(isAdmin(msg.sender) == true,Administrable: caller is not an admin) (contracts/Administrable.sol#11-14)
AumentWalletContract.proxyCallWithValue(bytes,address,uint256) (contracts/AumentWalletContract.sol#48-53) compares to a boolean constant:
        -success != true (contracts/AumentWalletContract.sol#50)
AumentWalletContract.proxyCallWithoutValue(bytes,address) (contracts/AumentWalletContract.sol#55-60) compares to a boolean constant:
        -success != true (contracts/AumentWalletContract.sol#57)
AumentWalletContract.onlyAdmin() (contracts/AumentWalletContract.sol#18-25) compares to a boolean constant:
        -require(bool,string)(aumentToken.isAdmin(msg.sender) == true,ERC20: caller is not an admin) (contracts/AumentWalletContract.sol#20-23)
EscrowFactoryProxy.deployAndCloseEscrow(bytes,bytes32,address,bool) (contracts/EscrowFactoryProxy.sol#28-70) compares to a boolean constant:
        -success != true (contracts/EscrowFactoryProxy.sol#65)
EscrowFactoryProxy.onlyAdmin() (contracts/EscrowFactoryProxy.sol#16-23) compares to a boolean constant:
        -require(bool,string)(aumentToken.isAdmin(msg.sender) == true,Escrow: caller is not an admin) (contracts/EscrowFactoryProxy.sol#18-21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Different versions of Solidity are used:
        - Version used: ['>=0.6.0<0.8.0', '^0.6.6']
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Pausable.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Pausable.sol#3)
        - ^0.6.6 (contracts/Administrable.sol#1)
        - ^0.6.6 (contracts/AumentWalletContract.sol#1)
        - ^0.6.6 (contracts/ERC20Aument.sol#1)
        - ^0.6.6 (contracts/EscrowContract.sol#1)
        - ^0.6.6 (contracts/EscrowFactoryProxy.sol#1)
        - ^0.6.6 (contracts/EscrowProxy.sol#1)
```



```
        - ^0.6.6 (contracts/IERC20Aument.sol#1)
        - ^0.6.6 (contracts/ProxyStorage.sol#1)
        - ^0.6.6 (contracts/RevertMsg.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20Pausable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Pausable.sol#3) is too complex
Pragma version^0.6.6 (contracts/Administrable.sol#1) allows old versions
Pragma version^0.6.6 (contracts/AumentWalletContract.sol#1) allows old versions
Pragma version^0.6.6 (contracts/ERC20Aument.sol#1) allows old versions
Pragma version^0.6.6 (contracts/EscrowContract.sol#1) allows old versions
Pragma version^0.6.6 (contracts/EscrowFactoryProxy.sol#1) allows old versions
Pragma version^0.6.6 (contracts/EscrowProxy.sol#1) allows old versions
Pragma version^0.6.6 (contracts/IERC20Aument.sol#1) allows old versions
Pragma version^0.6.6 (contracts/ProxyStorage.sol#1) allows old versions
Pragma version^0.6.6 (contracts/RevertMsg.sol#1) allows old versions
solc-0.6.6 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AumentWalletContract.proxyCallWithValue(bytes,address,uint256) (contracts/AumentWalletContract.sol#48-53):
        - (success,data) = address(targetAddress).call{value: value}(_data) (contracts/AumentWalletContract.sol#49)
Low level call in AumentWalletContract.proxyCallWithoutValue(bytes,address) (contracts/AumentWalletContract.sol#55-60):
        - (success,data) = address(targetAddress).call(_data) (contracts/AumentWalletContract.sol#56)
Low level call in EscrowFactoryProxy.deployAndCloseEscrow(bytes,bytes32,address,bool) (contracts/EscrowFactoryProxy.sol#28-70):
        - (success,data) = address(proxyAddress).call(payload) (contracts/EscrowFactoryProxy.sol#63)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

ERC20Aument (contracts/ERC20Aument.sol#8-201) should inherit from IERC20Aument (contracts/IERC20Aument.sol#6-10)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance

Variable Administrable._admins (contracts/Administrable.sol#4) is not in mixedCase
Parameter AumentWalletContract.proxyCallWithValue(bytes,address,uint256)._data (contracts/AumentWalletContract.sol#48) is not in mixedCase
Parameter AumentWalletContract.proxyCallWithoutValue(bytes,address)._data (contracts/AumentWalletContract.sol#55) is not in mixedCase
Parameter ERC20Aument.getSigner(bytes32,bytes)._hash (contracts/ERC20Aument.sol#151) is not in mixedCase
Parameter ERC20Aument.getSigner(bytes32,bytes)._signature (contracts/ERC20Aument.sol#151) is not in mixedCase
Constant ERC20Aument.precision (contracts/ERC20Aument.sol#13) is not in UPPER_CASE_WITH_UNDERSCORES
Variable ProxyStorage._escrowImplementationAddress (contracts/ProxyStorage.sol#4) is not in mixedCase
Variable ProxyStorage._erc20AumentContractAddress (contracts/ProxyStorage.sol#5) is not in mixedCase
Variable ProxyStorage._escrowFactoryAddress (contracts/ProxyStorage.sol#6) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (node_modules/@openzeppelin/contracts/utils/Context.sol#21)" inContext (node_modules/@openzeppelin/contracts/utils/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

ProxyStorage._escrowImplementationAddress (contracts/ProxyStorage.sol#4) is never used in EscrowContract (contracts/EscrowContract.sol#6-36)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#64-66)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#72-74)
```



```
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#89-91)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#103-105)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#134-137)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#170-173)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#189-192)
removeAdmin(address) should be declared external:
        - Administrable.removeAdmin(address) (contracts/Administrable.sol#27-31)
withdrawToken(address,uint256) should be declared external:
        - AumentWalletContract.withdrawToken(address,uint256) (contracts/AumentWalletContract.sol#33-37)
withdrawEther(address,uint256) should be declared external:
        - AumentWalletContract.withdrawEther(address,uint256) (contracts/AumentWalletContract.sol#39-46)
addWhitelistAddress(address) should be declared external:
        - ERC20Aument.addWhitelistAddress(address) (contracts/ERC20Aument.sol#49-52)
removeWhitelistedAddress(address) should be declared external:
        - ERC20Aument.removeWhitelistedAddress(address) (contracts/ERC20Aument.sol#54-57)
mint(address,uint256) should be declared external:
        - ERC20Aument.mint(address,uint256) (contracts/ERC20Aument.sol#78-81)
setTransferFee(uint256) should be declared external:
        - ERC20Aument.setTransferFee(uint256) (contracts/ERC20Aument.sol#95-99)
metaTransfer(bytes,address,uint256,uint256) should be declared external:
        - ERC20Aument.metaTransfer(bytes,address,uint256,uint256) (contracts/ERC20Aument.sol#137-145)
deployAndCloseEscrow(bytes,bytes32,address,bool) should be declared external:
        - EscrowFactoryProxy.deployAndCloseEscrow(bytes,bytes32,address,bool) (contracts/EscrowFactoryProxy.sol#28-70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
. analyzed (18 contracts with 78 detectors), 77 result(s) found
```

Note to Users

The audit does not cover out-of-scope functionality such as Loan Requests - approvals and denials, validity periods, checks for repayment, and addition to the whitelist array from the frontend.

Summary

In this report, we have considered the security of Aument. We performed our audit according to the procedure described above.

Some Issues of high, medium, low and informational severity were found during the course of the audit.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Aument Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Aument Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2023

For

AUMENT

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com