DeFiesta

# Aument

SECURITY REVIEW

Date: 19 March 2023

# CONTENTS

## 1. About Defiesta

We are Defiesta – a company on a mission to make web3 protocols more secure, cost-efficient and user-friendly. Our team boasts extensive experience in the web3 space as both smart contract auditors and developers that have worked on top 100 blockchain projects with multi-million dollars in market capitalization.

Book an audit and learn more about us at defiesta.net or @DefiestaAudits

## 2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Defiesta against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

## 3. About Aument

AUMENT AG is a Swiss company that goes hand in hand with the digitalization and technological progress and aims at providing high-quality financial services to its clients. AUMENT AG is the continuity of a long-term vision, bringing the financial services to the new technological contemporaneity.

Switzerland is known as a financial center, and Zug, in particular, is recognised as a world's "Crypto Valley". Therefore, AUMENT AG is established in the heart of the friendly environment with the aim to integrate into the Swiss and the international financial market and local crypto-friendly community while applying FINMA legal standards and compliance requirements.

## 4. Risk Classification

| **Severity** | **Impact: High** | **Impact: Medium** | **Impact: Low** |
|---|---|---|---|
| **Likelihood: High** | Critical | High | Medium |
| **Likelihood: Medium** | High | Medium | Low |
| **Likelihood: Low** | Medium | Low | Low |

### 4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** - results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** - losses will be limited but bearable - and covers vectors similar to griefing attacks that can be easily repaired.

### 4.2 Likelihood

- **High** - almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** - still relatively likely, although only conditionally possible
- **Low** - requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

## 5. Audit Summary

The Augment 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.

The Nat Spec is comprehensive. The code's readability could be further improved via the implementation of the Informational findings (not included in the report), which also outline some foundational best practices.

We extend our gratitude to the Blue Trading's blockchain team for their exemplary responsiveness, offering comprehensive clarifications and detailed responses to our inquiries.

We would also like to point out that the project's network of choice - IOTA's Shimmer, is still a relatively unexplored territory in terms of performance and network-level bugs and issues that might create additional attack surfaces.

### 5.1 Protocol Summary

| | |
|---|---|
| **Project Name** | **Aument** |
| **Repository** | Aument |
| **Type of Project** | Loans |
| **Audit Timeline** | 10 days |
| **Review Commit Hash** | 03573264991c49eb4608fdca9cef7ded9fa44cf9 |
| **Fixes Review Commit Hash** | N/A |

### 5.2 Scope

The following smart contracts were in the scope of audit:

| **File** | **nSLOC** |
|---|---|
| contracts/ AumentWalletContract.sol | 243 |
| contracts/Administrable.sol | 77 |

|  |  |
|---|---|
| Contract/ERC20Aument.sol | 47 |
| Contract/EscrowContract.sol | 376 |
| Contract/EscrowFactoryProxy.sol | 88 |
| Contract/EscrowProxy.sol | 149 |
| Contract/ProxyStorage.sol | 50 |
| Contract/RevertMsg.sol | 22 |
| **Total** | **1052** |

## 6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical** and **High** issues: **0**
- **Medium** issues: **3**
- **Low** issues: **3**

| ID | Title | Severity |
|---|---|---|
| [H-01] | Reentrancy risk in AumentWalletContract.sol | High |
| [H-02] | Centralization risk in ERC20Aument.sol | High |
| [M-01] | Missing input validation checks in AumentWalletContract.sol | Medium |
| [M-02] | Missing input validation checks in setAumentWalletAddress( ) in ERC20Aument.sol | Medium |
| [M-03] | Contract creation dependance in EscrowContract.sol | Medium |
| [M-04] | Inaccurate code behaviour in EscrowProxy.sol | Medium |
| [M-05] | Missing input validation checks in EscrowProxy.sol | Medium |

## 7. Findings

### [H-01] Reentrancy risk in AumentWalletContract.sol

**Severity** High

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.

**Recommendations**

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

**Team Response**

Resolved.

### [H-02] Centralization risk in ERC20Aument.sol

**Severity** High

Risk

**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.

**Recommendations**

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

**Team Response**

Acknowledged.

## [M-01] Missing input validation checks in AumentWalletContract.sol

### Severity

Medium Risk

### Description

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

### Recommendations

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.

### Team Response

Resolved.

## [M-02] Missing input validation checks in setAumentWalletAddress( ) in ERC20Aument.sol

### Severity

Medium Risk

### 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.

### Recommendations

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.

### Team Response

Resolved.

## [M-03] Contract creation dependance in EscrowContract.sol

### Severity

Medium Risk

### 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.

**Recommendations**

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

**Team Response**

Acknowledged.

## [M-04] Inaccurate code behaviour in EscrowProxy.sol

**Severity**

Medium Risk

**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().

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

**Recommendations**

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

**Team Response**

Resolved.

## [M-05] Missing input validation checks in EscrowProxy.sol

**Severity**

Medium Risk

**Description**

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

**Recommendations**

Consider implementing address checks, possible using require statements.

**Team Response**

Resolved.