Blaize.Security

LOCKON

LOCKON FINANCE
SMART CONTRACT AUDIT

November 15th, 2024 / V. 1.1

BLAIZE.SECURITY
WE SECURE WEB3 ECOSYSTEMS

Blaize.Security is a world-class web3 security provider that works with up-to-date technologies to build a safe environment for entire ecosystems. We provide security services for over 25 chains and ecosystems, and the list grows.

Blaize.Security has over 6 years in the web3 industry and offers a dedicated team of over 25 certified Security Researchers and Auditors to cover all platform components. Therefore we have separate teams for each security direction and service.

Pavlo Horbonos, Head of Security in Blaize

Software Engineer and Security Researcher with decades of experience in decentralized technologies, fintech, low-level programming, cyber + web3 security, and AI and Data Science. Certified auditor and security researcher.

https://www.linkedin.com/in/pavelmidvel/
https://twitter.com/MidvelCorp

Check more info on
https://security.blaize.tech

X (ex-Twitter): **https://x.com/BlaizeSecurity**

Linkedin: **https://www.linkedin.com/company/blaize-security**

Contact us:
security@blaize.tech

Table of Contents

Executive Summary

During the audit, we examined the security of smart contracts for the Lockon finance protocol. Our task was to find and describe any security issues in the platform's smart contracts. This report presents the findings of the security audit of the **Lockon** smart contracts conducted between **November 5th, 2024, and November 13th, 2024.** Note that this audit iteration is an additional review conducted on top of the protocol previously audited by the Blaize team on **August 29th, 2023**. The current audit focuses on the new contracts.

Blaize Security conducted comprehensive testing during the audit to verify the PositionUnitAdjusterModule contract, which utilizes several dependencies to extend its functionality. Some of these dependencies, such as SetToken.sol, Controller.sol, and ExtendModuleBase.sol, were already audited as part of the prior audit cycle. As these dependencies have not been modified in this iteration, they have not been re-audited in this round. However, we have identified two new contracts, Position.sol, and PreciseUnitMath.sol, that have been added to the codebase. The audit examined the module's primary operations, specifically unit adjustments for SetToken components, and evaluated scenarios in which units are recalculated based on actual balances. The module's logic allows only upward adjustments of token units to prevent unauthorized reduction of positions, ensuring that SetToken's components align with actual holdings.

The security team currently evaluates the project as **Highly Secure**, with few notes regarding the optimization of the contract.

Some notes and concerns from the security team:

1. This audit iteration is an additional review conducted on top of the protocol that was previously audited by the Blaize team on August 29th, 2023. The current audit focuses on the new contracts.

2. PositionUnitAdjusterModule contract utilizes several dependencies to extend its functionality. Some of these dependencies—such as SetToken.sol, Controller.sol, and ExtendModuleBase.sol—were already audited as part of the prior audit cycle. As these dependencies have not been modified in this iteration, they have not been re-audited in this round. However, we have identified two new contracts, Position.sol and PreciseUnitMath.sol, that have been added to the codebase.

3. For the contract to function correctly after deployment, it is crucial that the initialize() function is called. This function must be invoked before any other contract methods can be used. To ensure proper functionality, the deployment script should be modified to include this initialization step. Alternatively, after deployment, it is essential to manually verify that the initialize() function has been executed before any other contract interactions are triggered.

ADDITIONAL TOOLS USED

1) During the testing stage, auditors used **Hardhat framework** for additional unit tests suite and **Fourdry framework** for additional hypothesis and fuzzy testing.
2) For the manual and exploratory testing round the team prepared local deployment
3) Auditors used the output of Slither automated tool and conducted the interpretation of its results.

Auditing strategy and Techniques applied/Procedure

Blaize.Security auditors start the audit by developing an auditing strategy - an individual plan where the team plans methods, techniques, approaches for the audited components. That includes a list of activities:

MANUAL AUDIT STAGE

- Manual line-by-line code by at least 2 security auditors with crosschecks and validation from the security lead;
- Protocol decomposition and components analysis with building an interaction scheme, depicting internal flows between the components and sequence diagrams;
- Business logic inspection for potential loopholes, deadlocks, backdoors;
- Math operations and calculations analysis, formula modeling;
- Access control review, roles structure, analysis of user and admin capabilities and behavior;
- Review of dependencies, 3rd parties, and integrations;
- Review with automated tools and static analysis;
- Vulnerabilities analysis against several checklists, including internal Blaize.Security checklist;
- Storage usage review;
- Gas (or tx weight or cross-contract calls or another analog) optimization;
- Code quality, documentation, and consistency review.

and a wide spectrum of other vulnerable areas.

FOR ADVANCED COMPONENTS:

- Cryptographical elements and keys storage/usage audit (if applicable);
- Review against OWASP recommendations (if applicable);
- Blockchain interacting components and transactions flow (if applicable);
- Review against CCSSA (C4) checklist and recommendations (if applicable);

TESTING STAGE:

- Development of edge cases based on manual stage results for false positives validation;
- Integration tests for checking connections with 3rd parties;
- Manual exploratory tests over the locally deployed protocol;
- Checking the existing set of tests and performing additional unit testing;
- Fuzzy and mutation tests (by request or necessity);
- End-to-end testing of complex systems;

In case of any issues found during audit activities, the team provides detailed recommendations for all findings.

POST-AUDIT STEPS RECOMMENDED

To ensure the security of the contract, the **Blaize.Security** team suggests that the team follow post-audit steps:

1. Request audits of other protocol components (dApp, backend, wallet, blockchain, etc) from Blaize Security
2. Request consulting and deployment overwatch services provided by Blaize Security
3. Launch active protection over the deployed contracts to have a system of early detection and alerts for malicious activity. We recommend the AI-powered threat prevention platform **VigiLens**, by the **CyVers** team.
4. Launch a **bug bounty program** to encourage further active analysis of the smart contracts.
5. Request post-deployment assessment service provided by Blaize Security to ensure the correctness of the configuration, settings, cross-connections of deployed entities and live functioning

Audit Rating

Score: **10** /10

	RATING
Security	10
Logic optimization	10
Code quality	10
Testing suite	9.9
Documentation	10

Security: General mark for the security of the protocol.
The main mark for the audit qualification.

Logic optimization: Evaluation of how optimal the implementation is, including presence of extra/unused code, uncovered/extra cases, gas (or its analog) optimization, memory management optimization, etc
Code quality: Evaluation of best practices followed, code readability, structure and convenience of further development
Testing suite: Availability of the native tests suite, level of logic coverage, checks of critical areas being covered.
Documentation: Availability and quality of the documentation, coverage of core functionality and user flows: whitepaper, gitbook, readme, specs, natspec, comments in the code and other possible forms of documentation.

SECURITY RATING CALCULATION

Approximate weight of unresolved issues.
Critical: -3 points
High: -2 points
Medium: -0.5 points
Low: -0.1 points
Informational: -0.1 point (in general, depends on the context)

Note: additional concerns, violated checklist items (including standard vulnerabilities), and verified backdoors may influence the final mark and weight of certain issues.

Starting with a perfect score of 10:

Critical issues: 0 issue (0 resolved): 0 points deducted

High issues: 0 issues (0 resolved): 0 points deducted.

Medium issues: 0 issue (0 resolved): 0 points deducted

Low issues: 2 issues (1 resoled, 1 verified): 0 points deducted

Informational issues: 1 issues (1 resolved): 0 points deducted

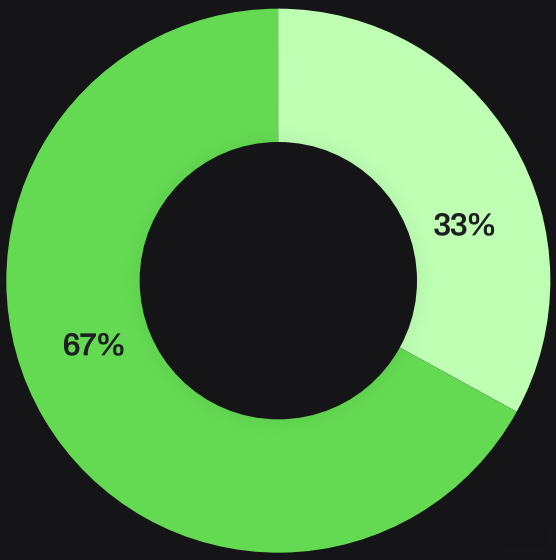
Other: 0 points deducted.

Security rating = 10

Technical Summary

THE GRAPH OF VULNERABILITIES DISTRIBUTION:

- Critical
- High
- Medium
- Low
- Info



The table below shows the number of the detected issues and their severity. A total of 3 problems were found. 3 issues were fixed or verified by the Customer's team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	0	0
Medium	0	0
Low	2	2
Info	1	1

Best practices and optimizations

- 1 items resolved

Section is marked as **resolved**

SEVERITY DEFINITION

CRITICAL

The system contains several issues ranked as very seriousand dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

HIGH

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

MEDIUM

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

LOW

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

INFO

The issue has no impact on the contract's ability to operate, yet is relevant for best practices. Or this status can be assigned to the issues related to the suspicious activity or substandard business logic decisions which cannot be classified without the comments from the team (and can be re-classified on the later audit stages).

Issues reviewed by the team can get the next statuses:
Resolved: issue is resolved by an appropriate patch or changes in the business logic
Verified: the team provided sufficient evidences that the issue describes desired behavior
Unresolved: neither path nor comments provided by the team, or they are not sufficient to resolve the issue
Acknowledged: the team accepts the misbehavior and connected risks

Audit Scope

Language/Technology: **Solidity**

Blockchain: **Ethereum**

The scope of the project includes:

- contracts/protocol/modules/v1/PositionUnitAdjusterModule.sol
- contracts/protocol/lib/Position.sol
- contracts/lib/PreciseUnitMath.sol

Repository: https://github.com/lockon-finance/core-contracts

The source code of the smart contract was taken from the branch:
main

Initial commit:

- 859d9af5b13f92e6ed5d995ad0a0d9134e56d2a6

During the audit process, code fixes occurred in the branch:
main

Final commit:

- 843923082f4f4418de612c9c079eed81ad9858f1

Protocol overview

DESCRIPTION

The PositionUnitAdjusterModule is a smart contract designed for use with SetTokens. It allows for the creation and management of tokenized baskets of assets (SetTokens), which are ERC20 tokens representing a collection of underlying assets. This module is specifically created to adjust the unit numbers (i.e., the amount per SetToken) of the components (underlying assets) within a SetToken to reflect the actual balances held, thereby correcting any discrepancies. The contract is not upgradable and does not store any ERC20 tokens.

ACCESS CONTROL

- Manager: The address that has control over the SetToken and can perform privileged actions.
- Operator: An authorized address that can perform certain actions, such as adjusting position units, on behalf of the manager.

VALUABLE ASSETS

The contract is not supposed to hold assets of any kind, thus any tokens sent directly to the contract will be locked.

MAIN FUNCTIONS

Initialize:
- Set up the module for a specific SetToken.
- Can only be called by the Manager.
- Registers the module with the SetToken.

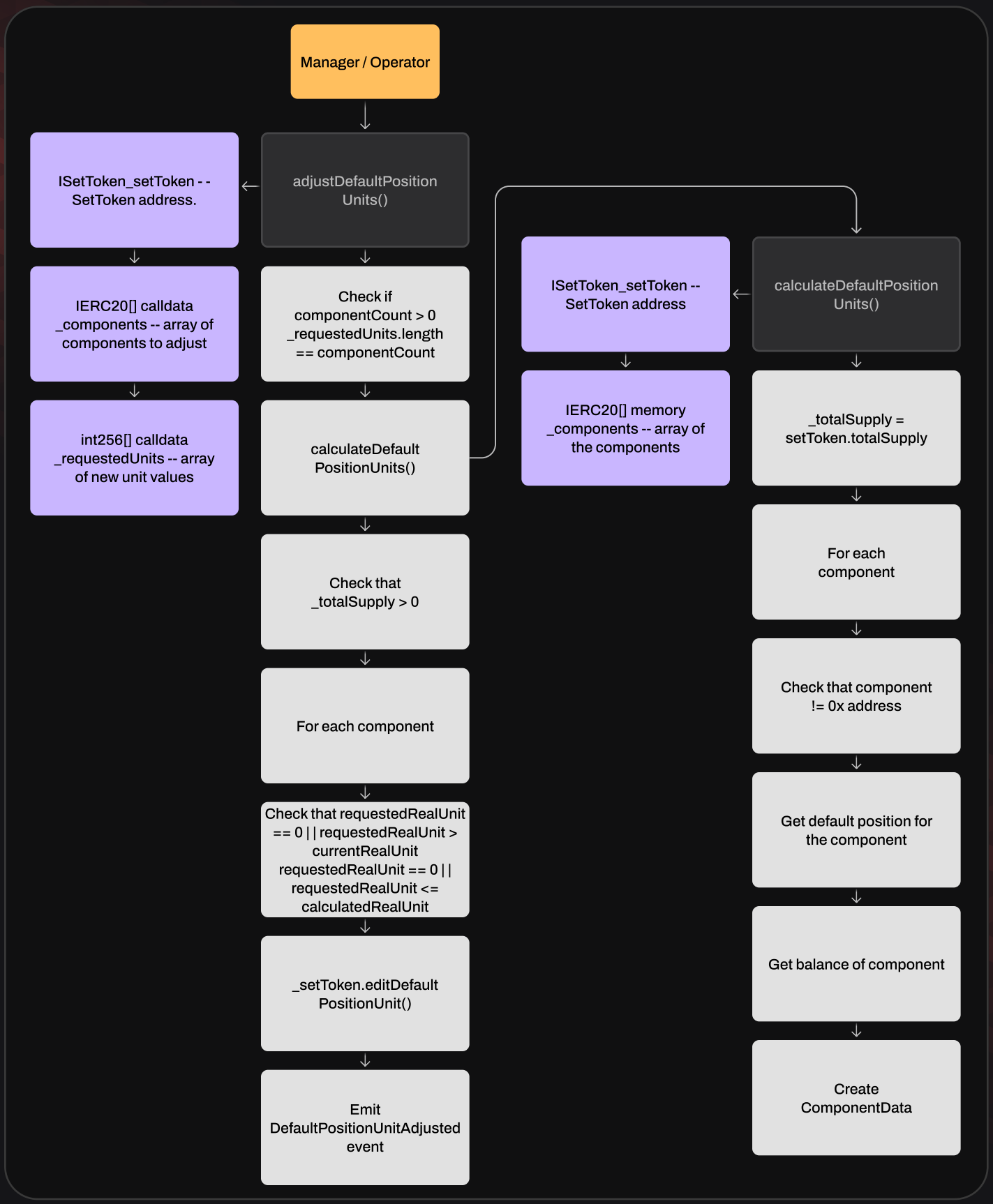
calculateDefaultPositionUnits:
- A view function that calculates the current and new (calculated) units for specified components.
- Returns the total supply of the SetToken and an array of ComponentData.

adjustDefaultPositionUnits:
- Adjusts the default position units of specified components in the SetToken.
- Can only be called by the manager or an authorized operator.
- Ensures that the adjustments are within acceptable bounds.

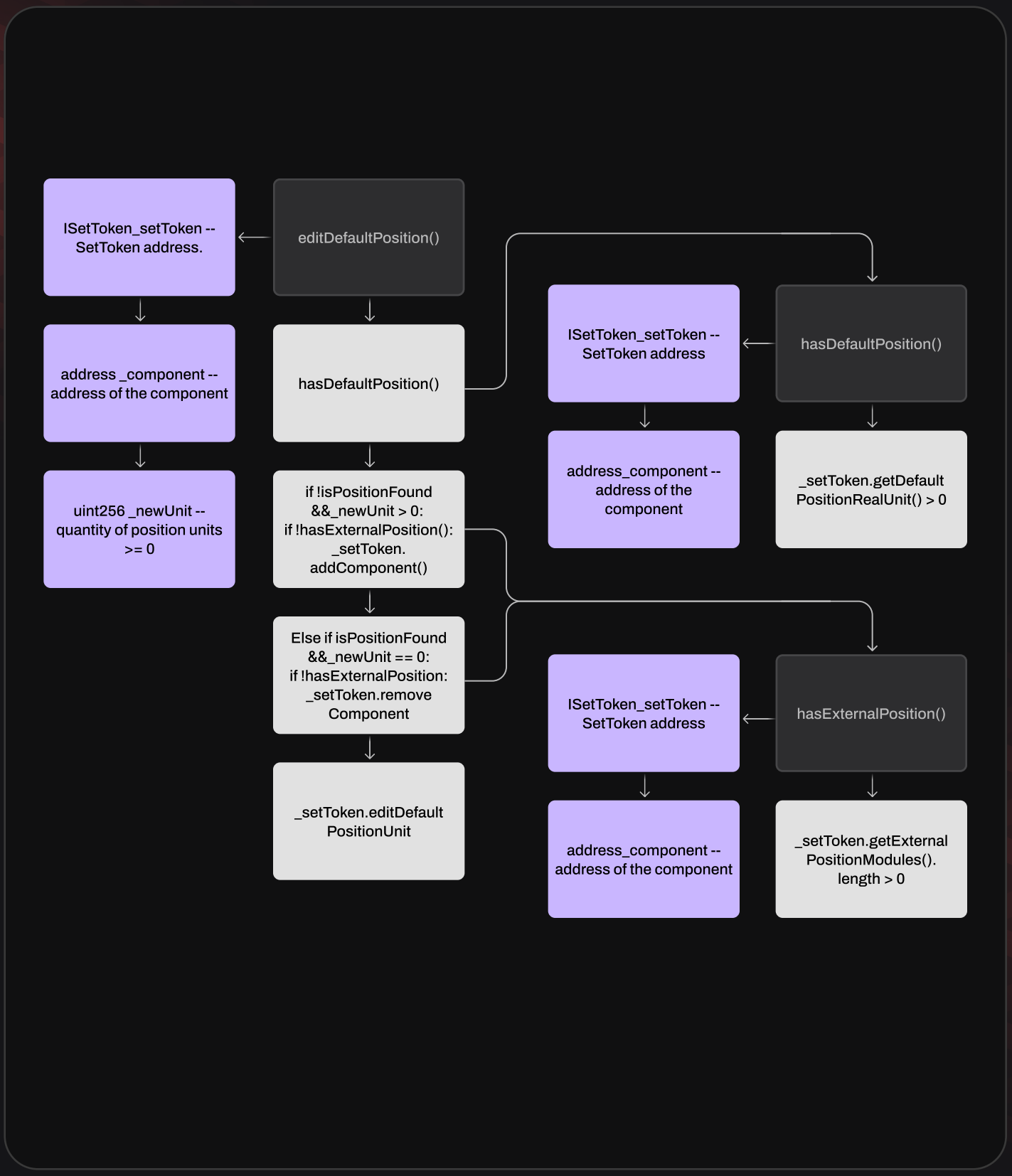
DEPLOYMENT

Deploy script located at utils/deploys/deployModules.ts -> function deployPositionUnitAdjusterModule() which gets controller and operator as function arguments.

POSITIONUNITADJUSTERMODULE.SOL



POSITION.SOL (WRAPPER FOR SETTOKEN FUNCTIONS)



Complete Analysis

STANDARD CHECKLIST / VULNERABLE AREAS

Storage structure and data modification flow	Pass
Access control structure, roles existing in the system	Pass
Public interface and restrictions based on the roles system	Pass
General Denial Of Service (DOS)	N/A
Entropy Illusion (Lack of Randomness)	N/A
Order-dependency and time-dependency of operations	Pass
Accuracy loss, incorrect math/formulas other violated operations with numbers	Pass
Validation of function parameters, inputs validation	Pass
Asset management, funds flow and assets conversions	N/A
Signatures replay and multisig schemes security	N/A
Asset Security (backdoors connected to underlying assets)	Pass
Incorrect minting, initial supply or other conditions for assets issuance	Pass
Global settings mis-using, incorrect default values	Pass
Violated communication between components/modules, broken co-dependencies	N/A
3rd party dependencies, used libraries and packages structure	Pass
Single point of failure	Pass
Centralization risk	Pass
General code structure checks and correspondence to best practices	Pass
Language-specific checks	Pass

LOW-1 Verified

COMPONENTS MIGHT REPEAT.

PositionUnitAdjusterModule.sol, adjustDefaultPositionUnits().
The function processes each element from the array `_components` inside the loop. However, there is no validation that the array `_components` contains a set of unique addresses inside the function. While it doesn't lead to any potential threat, it might be confusing for the manager who might accidentally put the same component in the loop. As a result, the manager might think that different components were specified, while the actual number of processed components may differ. Also, due to this, the component might end up with the wrong requested unit processed which was meant for another component.

Thus, the issue itself doesn't lead to security threats, however, it might still have some impact due to human factors or, in case the manager or operator who is supposed to execute function adjustDefaultPositionUnits() is a back-end worker, have impact due to issues in the code of the back-end.

RECOMMENDATION:

Validate that the array `_components` consists of unique elements. In case the function adjustDefaultPositionUnits() is supposed to be called by the back-end, consider including this validation in its code.

POST-AUDIT:

Lockon team verified that they will not perform duplicate checks for the following reasons.
1. The process of calling it systematically for operation has been verified off-chain.
2. Even if a duplicate component address is passed, there will be no problem because it can only be updated within the range of currentRealUnit and calculatedRealUnit).

While the Lockon team has verified that the correctness of input parameters will be verified off-chain, there is still a concern regarding absence of such validations directly in smart contract.

LOW-2 Resolved

LIBRARY "POSITION" IS IMPORTED AND NEVER USED.

PositionUnitAdjusterModule.sol
The library Position.sol is imported and its usage is declared at the line 31, however, its functions are never called in the code. Such logic might be confusing as it is implied that the library, which is declared, is supposed to be used in the code.

RECOMMENDATION:

Add the usage of the library's functions or remove the library. Also, validate if the current logic is correct at the line 121

```
int256 currentUnit =
_setToken.getDefaultPositionRealUnit(address(component));
```

and at the line 184

```
_setToken.editDefaultPositionUnit(address(component),
requestedRealUnit);
```

where functions of SetToken.sol are called directly instead of using functions from the library.

POST-AUDIT:

Added Position.sol library functionality by calling editDefaultPosition instead of editDefaultPositionUnit.

INFO-1 Resolved

FUNCTION ARGUMENT IS INT256 TYPE

PositionUnitAdjusterModule.sol: adjustDefaultPositionUnits(), variable _requestedUnits
Function argument _requestedUnits in adjustDefaultPositionUnits() function is int256 type, but the comment does not explain if this variable could be a negative value. As from the SetToken code comment: `Each unit is the # of components per 10^18 of a SetToken`, we assume that value could only be positive or 0. Additional checks should be added in case the value could not be negative.

RECOMMENDATION:

Verify if the function argument can be a negative value. Add a comment in the code explaining possible values of the argument (including whether it can or cannot be negative).

POST-AUDIT:

The Lockon Finance team has added a validation for negative values.

Resolved

BEST PRACTICES AND CODE STYLE VIOLATIONS, OPTIMIZATIONS

1. Variables initialization inside the loop.

PositionUnitAdjusterModule.sol
calculateDefaultPositionUnits(), lines 118, 121-123.
adjustDefaultPositionUnits(), lines 168-171.

Several variables are re-initialized inside the loop, increasing the gas consumed by the execution of the function. It is recommended to define variable outside of the loop.

RECOMMENDATION:

Consider correcting listed issue to increase code readability and logic optimization.

POST-AUDIT:

Variables were moved outside of the loop.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM

PositionUnitAdjuster module

PositionUnitAdjusterModule integration

- ✓ Should revert with OnlyManagerOrOperator (112ms)
- ✓ Change unit for first component (180ms)
- ✓ Change unit for second component (123ms)
- ✓ Should revert with OnlyManagerOrOperator (75ms)
- ✓ Revert if real units is negative number (119ms)
- ✓ Revert if wrong component is used (86ms)
- ✓ Revert if wrong length of arguments or no components (94ms)
- ✓ Revert if setComponent supply is 0 (81ms)
- ✓ Change unit for first component in suitable range (166m)

PositionUnitAdjuster module fuzz test

- ✓ [PASS] testFuzzAdjust(uint256,uint256) (runs: 266, μ: 237279, ~: 232097)

RESULTING COVERAGE (BY BLAIZE TEAM)

FILE	% STMTS	% BRANCH	% FUNCS
PositionUnitAdjusterModule.sol	100	82.14	97.3

Blaize Security team has additionally implemented a fuzzy test to ensure requested units are set only within appropriate range limited by invariants of PositionUnitAdjusterModule smart contract.

NATIVE TESTS OVERVIEW

The Lockon Finance team has provided their own set of unit tests with sufficient coverage and testing of some of edge-cases (For example, the case when same component is passed multiple time within a single execution of adjustDefaultPositionUnits()).

The coverage of Lockon Finance's native tests is 100%.

Disclaimer

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.