SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Alacrity LSD Protocol
Date: Aug 18, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Alacrity LSD Protocol
Approved By	Paul Fomichov | Lead Solidity SC Auditor at Hacken OU
Tags	ERC20 token; Staking
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://linktr.ee/alacritylsd
Changelog	23.06.2023 - Initial Review 21.07.2023 - Second Review 18.08.2023 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Alacrity LSD Protocol (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

The Alacrity LSD Protocol audit scope consists of a staking and rewarding protocol that accepts ERC20 and native token staking for stakers, and distributes rewards according to their staked values. Both the staking token and reward token can be configured, and different taking pools can be managed by admin addresses.
The core contract of the system are the following;

- *StakingPool.sol* - The staking contract for a single ERC20 token staking and reward token.
- *StakingPoolFactory.sol* - The contract that handles creation and management of several staking pools, including native token staking.
- *EthStakingPool.sol* - The staking pool contract for the native token.
- *CurrencyTransferLib.sol* - The helper library for transferring native tokens.
- *RewardsDistributionRecipient.sol* - Helper extendable contract to allow calls restricted to rewards distributor.
- *IVEALSD.sol* - Interface for the vote escrow token of Alacrity.
- *IALSD.sol* - Interface for the platforms native token.
- *IWETH.sol* - Interface for native token wrapper.
- *IStakingPool.sol* - Interface for the Staking Pool.

Privileged roles

- **Owner**: Can interact with the StakingPoolFactory.sol and use its privileged functions, such as deploying new pools, withdrawing excess rewards, and adding rewards to pools.
- **rewardsDistribution**: This is the StakingPoolFactory.sol contract. It can trigger privileged functions in the staking pool contracts, such as changing reward rate by notifying reward amount and withdrawing excess rewards.

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **7** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- Development environment description is not provided.
- NatSpec is not sufficient.

Code quality

The total Code Quality score is **7** out of **10**.

- Solidity style guides are not followed.
- Missing zero address checks.
- The Development Environment was not configured.

Test coverage

Code coverage of the project is **0%** (branch coverage).

- Tests were not provided.

Security score

As a result of the audit, the code contains **1** medium and **2** low severity issues. The security score is **9** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **2.0**.

The system users should acknowledge all the risks summed up in the risks section of the report.

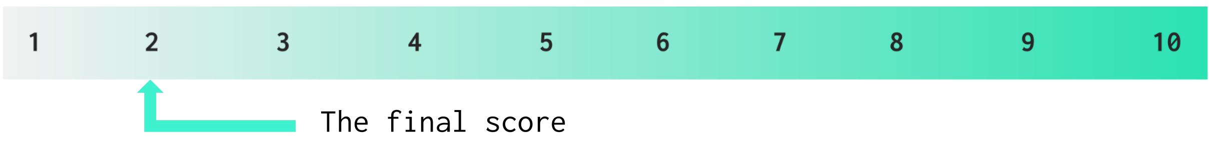

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
23 June 2023	4	3	2	3
21 July 2023	4	1	2	2
18 August 2023	2	1	0	0

Risks

- There are out of scope external calls made in the StakingPoolFactory.sol contracts' addRewards function. The security of the functionality cannot be verified.
- Users cannot get their rewards from the system unless the owner adds rewards via StakingPoolFactory contract. The timing and quantity of reward additions are determined by a centralized mechanism.
- The ALSD and veALSD contracts are not part of the audit scope, and their security cannot be verified.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Passed	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Passed	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. Contracts shouldn't rely on values that can be changed in the same transaction.	Not Relevant	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Not Relevant	
Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed	
Style Guide Violation	Style guides and best practices should be followed.	Failed	I07
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

■■■■ Critical

C01. Invalid Calculations

Impact	High
Likelihood	High

The refund functionality in the EthStakingPool contract is not working correctly when the fee for deposits is enabled.

The user is refunded twice, first in the *_transferStakingTokenFee()* function and second in the *_transferStakingToken()*.

In both cases, the refunded amount is calculated incorrectly, but most importantly, the refund functionality is causing double spending.

A malicious actor can send a *stake()* transaction with a low *amount* and a high *msg.value* to drain native tokens from the contract.

He will be refunded twice based on the *msg.value - fee* and *msg.value - amount* calculations in both functions, leading to an amount of native tokens almost equal to msg.value being extracted.

Path:
./EthStakingPool.sol : _transferStakingTokenFee(), _transferStakingToken()

Recommendation: Fix the double refund problem or revert when msg.value is not equal to the amount in stake() function.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

C02. Data Inconsistency

Impact	High
Likelihood	High

Transfer direction is incorrect in the *_transferStakingTokenFee* function. Fee is getting stuck in the *ETHStakingPool* contract and not getting sent to the fee manager's address.

This will lead the fee manager to not receive staking fee payments.

Path:
./EthStakingPool.sol : _transferStakingTokenFee()

Recommendation: Transfer the received WETH after depositing, to the feeManager address.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

■■■ High

H01. Contradiction In Function Name

Impact	Medium
Likelihood	High

Although the *withdrawRewards()* function name in the *StakingPool.sol* contract indicates that it will withdraw the rewards, it actually allows the owner to withdraw the stakingToken in the contract that is excess from user stakes.

The *withdrawRewards()* function name in the *EthStakingPool.sol* contract is redundant since the funds are stored as WETH and not native tokens that are being transferred.

This may lead to unexpected behavior.

Paths:
./EthStakingPool.sol : withdrawRewards()
./StakingPool.sol : withdrawRewards()

Recommendation: Describe the functionality and rename the functions to express its code.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

H02. Funds Lock

Impact	Medium
Likelihood	High

In the *stake()* function of the *StakingPool* contract, there is a *payable* modifier used, but there is no check on the msg.value.

If the transferred token is not native (in the case the interacted contract is not *EthStakingPool.sol*) but there is a transfer of native tokens (ETH), they will be locked in the contract.

Path:
./StakingPool.sol : stake()

Recommendation: Add a check in the *stake()* function so that when the staking token is not the native token, it should revert on msg.value > 0.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

Medium

M01. Missing Validation

Impact	Medium
Likelihood	Medium

In *deployPool* function, there is no check in case a pool is created for the same staking token.

This will cause all the info for a staking token and its pool to be lost and it will be impossible to add rewards to that pool.

This will lead to funds being locked.

Path:
./StakingPoolFactory.sol : deployPool()

Recommendation: Put a *require* statement that checks if a pool is already created for the given token address.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed

M02. Unverifiable Logic

Impact	Low
Likelihood	High

StakingPoolFactory contract makes external calls to the *ALSD* and *veALSD* contracts which are out of scope. Since their implementation cannot be verified, Hacken does not guarantee StakingPoolFactory safety.

Path:
./StakingPoolFactory.sol : addRewards()

Recommendation: Include the contracts in the scope or document their functionality in the flow in detail.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Mitigated (The Customer stated that the only minter will be the StakingPoolFactory contract for these tokens.)

M03. Denial of Service

Impact	Medium
Likelihood	Medium

In the *setFeeManager* function in the *StakingPool.sol* contract, Address parameter is being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0 and the stake function can revert (DoS) on transferFrom to address(0).

Path:
./StakingPool.sol : setFeeManager()

Recommendation: Implement zero address check in the function.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed

Low

L01. Missing Zero Address Validation

Impact	Low
Likelihood	Medium

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Paths:
./EthStakingPool.sol : constructor(), withdrawRewards()
./StakingPool.sol : constructor(), setFeeManager(), withdrawRewards()
./StakingPoolFactory.sol : constructor(), deployPool(), withdrawRewards(), addRewards()

Recommendation: Implement zero address checks.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Reported (addRewards function is missing the zero address validation)

L02. Empty Function

Impact	Low
Likelihood	Medium

receive function is used as empty and there is no explanation regarding why it is implemented. This may lead to confusion and misinterpretation about its purpose among developers/users, resulting in potential misuse, bugs, or unwanted behavior in the contract's execution.

Path:
./StakingPool.sol : receive

Recommendation: Put a message that explains why empty receive is used.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Reported (remains in StakingPool.sol. Not documented.)

L03. Floating Pragma

Impact	Low
Likelihood	Medium

Locking the pragma helps 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.

Paths:
./RewardsDistributionRecipient.sol
./StakingPool.sol
./StakingPoolFactory.sol
./EthStakingPool.sol
./lib/CurrencyTransferLib.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

L04. Missing Event Emitting

Impact	Low
Likelihood	Medium

Events for critical state changes should be emitted for tracking things off-chain. Some functions do not emit any event after making important state changes.

Paths:
./EthStakingPool.sol : withdrawRewards
./StakingPool.sol : stake, withdraw, getReward, setFeeManager, setDepositFee, notifyRewardAmount, setDepositFeeState,

withdrawRewards
./StakingPoolFactory.sol : withdrawRewards, addRewards

Recommendation: Emit event for critical state changes.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

Informational

I01. State Variables That Can Be Declared Immutable

Variables' StakingPool.rewardsDuration, StakingPool.rewardsToken, StakingPool.stakingToken, EthStakingPool.wethvalue, StakingPoolFactory.rewardsToken, StakingPoolFactory.nativeTokenWrapper, StakingPoolFactory.alsdContract, StakingPoolFactory.vealsdContract are set in the constructor.

These variables can be declared immutable.

This will lower the Gas taxes.

Paths:
./EthStakingPool.sol
./StakingPool.sol
./StakingPoolFactory.sol

Recommendation: Declare mentioned variables as immutable.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

I02. Redundant Use Of SafeMath

Since Solidity v0.8.0, the overflow/underflow check is implemented via ABIEncoderV2 on the language level - it adds the validation to the bytecode during compilation.

There is no need to use the SafeMath library.

Paths:
./EthStakingPool.sol
./StakingPool.sol
./StakingPoolFactory.sol

Recommendation: Remove the SafeMath library.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Reported (Revised commit: 204f265)

I03. Unused Functionality

The StakingPool.sol contract inherits the Ownable contract but never uses its functionality. Unused functionality leads to increasing deployment Gas price and decrease code quality.

Path:
./contracts/StakingPool.sol : Ownable

Recommendation: Remove unused functionality to save Gas on deployment and increase code quality.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

I04. Functions That Can Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared as external.

Paths:
./StakingPool.sol: withdraw(), getReward()
./StakingPoolFactory.sol: getStakingPoolAddress(), deployPool()

Recommendation: Declare the functions as external.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

I05. Redundant Require Statement

On line 8, require statement that compares two same number is redundant since it's meaningless.

Redundant declarations cause extra Gas consumption and decrease code readability.

Path:
./RewardsDistributionRecipient.sol : onlyRewardsDistribution()

Recommendation: Remove the require statement.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

I06. Unused Imports

Unused imports should be removed from the contracts. Unused imports are allowed in Solidity and do not pose a direct security issue. It is best practice to avoid them as they can decrease readability.

Path:
./StakingPool.sol : CurrencyTransferLib

Recommendation: Remove the redundant import.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

I07. Style Guide Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be ordered and grouped by their visibility as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)
- External functions
- Public functions
- Internal functions
- Private functions

Within each grouping, view and pure functions should be placed at the end.

Furthermore, following the Solidity naming convention and adding NatSpec annotations for all functions are strongly recommended. These measures aid in the comprehension of code and enhance overall code quality.

Paths:
./StakingPool.sol
./EthStakingPool.sol

Recommendation: Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts. Providing comprehensive NatSpec annotations for functions and following Solidity's naming conventions further enrich the quality of the code.

Follow the official Solidity guidelines.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Reported (Revised commit: 204f265)

I08. Redundant Declaration

uint256 variables are zero as default. Therefore, there is no need to assign a zero value to them during declaration.

address variables are zero as default. Therefore, there is no need to assign a zero address value to them during declaration.

Redundant declarations consume unnecessary Gas and decrease code readability.

Path:
./StakingPool.sol: periodFinish, rewardRate, feeManager

Recommendation: Remove the assigning.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

I09. Data Inconsistency

It is not clear or logical why the ERC20Upgradable's interface was used even though the tokens utilized in the system are not upgradeable. Reading this code could potentially cause confusion for users. It is important to ensure that the code and its various components align with their intended functionality, to minimize misunderstandings and ensure proper use.

Path:
./CurrencyTransferLib.sol

Recommendation: Use IERC20 to represent tokens.

Found in: 0c8faac784e9d366b309501f5b3f3eb550599974

Status: Fixed (Revised commit: 204f265)

I10. Hardcoded Value

In the StakingPool.stake() functions check for native token, the address of WETH is hardcoded.

Hardcoding values is against best practices and can decrease the reliability of the code.

Path: ./StakingPool.sol : stake()

Recommendation: Declare mentioned address as a variable and use that instead of hardcoded value.

Found in: 204f265d3447faa59cc9586614146e035ccf9ca3

Status: New

I11. Require Statement Without An Error Message

Require statements on line 32 in CurrencyTransferLib contract and on line 213 in StakingPool contract lack a proper error message. The absence of clear and informative error messages in these "require" statements can lead to confusion during debugging, maintenance, and error resolution processes.

Paths: ./StakingPool.sol: withdrawExcess()

./lib/CurrencyTransferLib.sol: safeTransferNativeToken()

Recommendation: Put proper error messages for require statements.

Found in: 204f265d3447faa59cc9586614146e035ccf9ca3

Status: New

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with 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).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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 recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

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

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/AlacrityLSD/smartcontracts
Commit	0c8faac784e9d366b309501f5b3f3eb550599974
Whitepaper	Link
Requirements	Link
Technical Requirements	-
Contracts	File: EthStakingPool.sol SHA3: c1ae318b4c7acf32d54a05df56f2a82b7817613645fa0be9d79a13d21db4fc5b File: RewardsDistributionRecipient.sol SHA3: 461e6bfd82d13fb834451583cab563252a56a56af854376a925dc666eac47184 File: StakingPool.sol SHA3: 9269bd75738ae8a96adb38fe06c72b5673b31289193e4e7119ae82ecd3e36a2b File: StakingPoolFactory.sol SHA3: 5b1e18086e87d79288740537a5b9e8ef9f0a34a9a957574522ae203cf51fca71 File: interfaces/IALSD.sol SHA3: 4df8b51155ec43cca450d90dd3a846e31853ea0a8e0ed2297d979049c2fa9648 File: interfaces/IStakingPool.sol SHA3: 9838faa10acada96aa32f7ee97120f4b3aa8919980520c3a00a9ef6643554ceb File: interfaces/IVEALSD.sol SHA3: 86570988a324ad91422bf5a9b2f35326014c3cc9c380cedd345d7f1c8fbaee2f File: interfaces/IWETH.sol SHA3: 6253e3d4eec3c5c3124bd0f3a231cfc2f54943790a1d9e5d2ee2eff413a4407f File: contracts/lib/CurrencyTransferLib.sol SHA3: 2dcc5df67834083c8b3e575f1c1c757088993c1a6f82c7cf119af3c90f96a254

Second review scope

Repository	-
Commit	-
Whitepaper	Link
Requirements	Link
Technical Requirements	-

Contracts	File: EthStakingPool.sol SHA3: e5839084608838556e48ed1472e060f8baaabf2287b093863db544adbddc9edf File: RewardsDistributionRecipient.sol SHA3: 707e0223e0e972163c7f43c45f44261b644122723e4707e62e18be65622c1345 File: StakingPool.sol SHA3: b447ec8958baf3e28ba1493f4449d7f22df761c9b689c26c2945db108dab5e54 File: StakingPoolFactory.sol SHA3: df6712f0e2b0a99db7d8a9435ff721e311af54eac32fcd0c89b291a7db85398b File: interfaces/IALSD.sol SHA3: 4df8b51155ec43cca450d90dd3a846e31853ea0a8e0ed2297d979049c2fa9648 File: interfaces/IStakingPool.sol SHA3: 92d895b16c2f202f945d1c15894eeb6f4e0e956b86a869b9fde6295da7bc7995 File: interfaces/IVEALSD.sol SHA3: 34ad030ba015e1fe10368efe4dd26008df5c1061fa6c4607fc8b4d73982d69b1 File: interfaces/IWETH.sol SHA3: f41508a3b0b4e7268a9aa4c752d2f74f2b61678178c1eade0e1c5950d2bf267f File: contracts/lib/CurrencyTransferLib.sol SHA3: 510c8ca0ddc8d31027a3ee511958c66e86c65c40a4dce6a6e4f8f623836696c6

Third review scope

Repository	https://github.com/AlacrityProtocolLSD/smartcontracts
Commit	204f265d3447faa59cc9586614146e035ccf9ca3
Whitepaper	Link
Requirements	Link
Technical Requirements	-
Contracts	File: EthStakingPool.sol SHA3: f2c49cfb7fe00c20b41eef41b1ed6ce011bd11c2cab24b827f4c97f56c5d5149 File: RewardsDistributionRecipient.sol SHA3: 7f2d0644cda152aff46ebec98dbb729332bd23269dcdff95ef31c6464ce45d70 File: StakingPool.sol SHA3: 98389339c779ebe1f9a09e04689ba66da791e3620f1d3ec6a560b9e0c07b40b9 File: StakingPoolFactory.sol SHA3: 86268d056d93e28b6b472f3592d956cd779d7f64f82c725019a65accbaf9dbc8 File: interfaces/IALSD.sol SHA3: a226979af274a0901ffa3748ab30427cfe644d2abc440c8189304dfbd6fe1c5f File: interfaces/IStakingPool.sol SHA3: d9864f935a7106b3e0ec693b47037289ad64d12df425e297dc9083a2ebc92526 File: interfaces/IVEALSD.sol SHA3: fe6c64321cc3ddbbaaff64f1843b5a13edaa2defe29431d6416fb76e9279a4e3

	File: interfaces/IWETH.sol SHA3: f2220c80fbd6236eb23cef1a661bf2db01e3bac92807bb437bbfb1af6a4b3d6d File: lib/CurrencyTransferLib.sol SHA3: 67056c7152028de64a8b56c67e3ea3f07923783329fdd68ed04b45e2f0ebd733