SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY

LyveFinance

AUDIT

SECURITY ASSESSMENT

08. August, 2023

FOR

Introduction

SolidProof.io is a brand of the officially registered company MAKE Network GmbH, based in Germany. We're mainly focused on Blockchain Security such as Smart Contract Audits and KYC verification for project teams.
Solidproof.io assess potential security issues in the smart contracts implementations, review for potential inconsistencies between the code base and the whitepaper/documentation, and provide suggestions for improvement.

Disclaimer

SolidProof.io reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyze.

Project Overview

Summary

Project Name	Lyve FInance
Website	https://www.lyvefi.xyz/
About the project	We will bridge most mainstream LSD tokens to l2 using LayerZero's ProxyOFT technology and directly use them for Lyve's Farming. As l2 matures and people pay more attention to security and privacy, we believe there will be more and more LSD projects supporting l2, and we will see more LSD projects and liquidity on l2
Chain	Linea
Language	Solidity
Codebase Link	https://github.com/LyveFinance/contract-lsd/tree/main/lsd/contracts
Commit	632f7bb
Unit Tests	Not Provided

Social Medias

Telegram	N/A
Twitter	https://twitter.com/LyveFinance
Facebook	N/A
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	N/A
Discord	N/A
Youtube	N/A
TikTok	N/A
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Changelog
v1.0	03. August 2023	• Layout Project • Automated- /Manual-Security Testing • Summary
v1.1	08. August 2023	• Reaudit

Note - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project. This analysis did not include functional testing (or unit testing) of the contract/s logic. We cannot guarantee 100% logical correctness of the contract as it was not functionally tested by us.

File Overview

The project provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

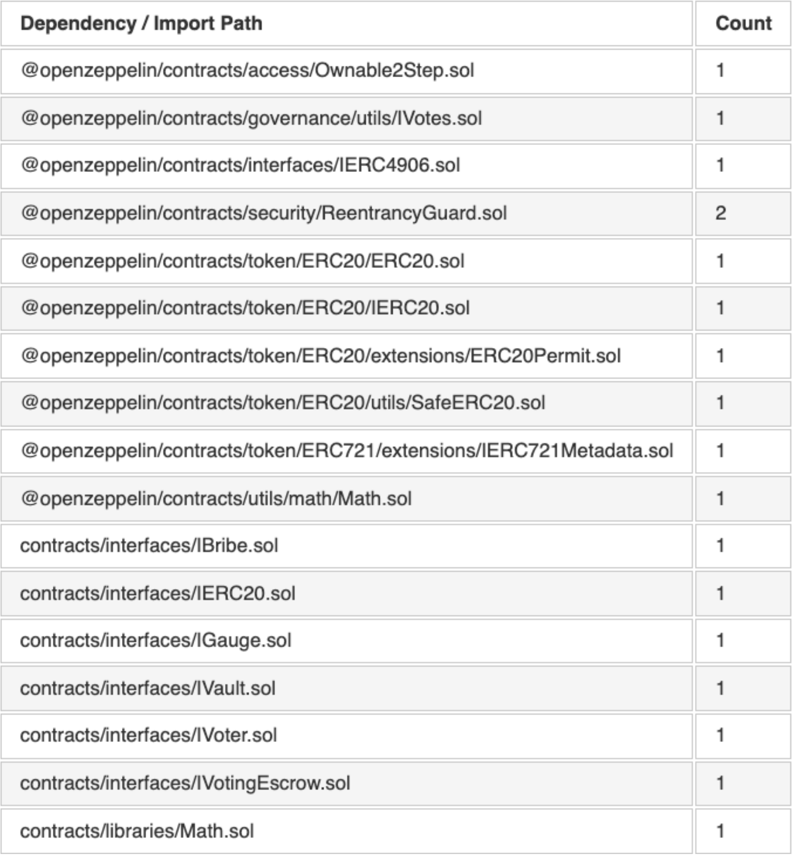
File Name	SHA-1 Hash
contracts/interfaces/ILsdRateOracle.sol	6d77d543f4fa841dda3b59c7b4c1310aa8639b78
contracts/interfaces/ILayerZeroUserApplicationConfig.sol	619d629e55c336e6297b51444c9751b25d699b7c
contracts/interfaces/IxF33dReceiver.sol	1a810ca1a0856102710b0595a038415ef6397d56
contracts/interfaces/IGauge.sol	2dcdbb5b18b8b4d1b71571f970c422517566e6aa
contracts/interfaces/ILayerZeroReceiver.sol	40fdfafbbd43411c04fab817e023e5162ab86386
contracts/interfaces/IVoter.sol	4d5d5b4c1286f2a5bca6377172d0b408fab80d12
contracts/interfaces/IVaultFactory.sol	37a9b9c77282568b8e1c1cada439814c78fc5a47
contracts/interfaces/AggregatorV3Interface.sol	5f3eff85813f2f28b9ae44fb86c82274e98e5cdd
contracts/interfaces/IVault.sol	608875320ae0f45ff4f8a661486a5646dbd62f08
contracts/interfaces/ILayerZeroEndpoint.sol	89e89b610c294d760037ba188a7a470a890a64fa
contracts/interfaces/IxF33dAdapter.sol	30ebfaeb455f03d433284bc42fe52e633973b4be
contracts/interfaces/IVotingEscrow.sol	5b58414c8d5cae855d26d3aa5c7e8a4e424807a5
contracts/xF33dReceiverChainlink.sol	0d13d4f4fea1870119bd7c9669049ccf50c1effb
contracts/xF33dAdapterChainlink.sol	51dcc01d03b16bff9336cb54704fce39bdac2e91
contracts/factories/VaultFactory.sol	49edb2bd49763c07c009787f968c4a0ed45498c1

contracts/xF33dReceiver.sol	190296da7adee584eacfeccba5803da8dea6c34f
contracts/VaultGauge.sol	594f87f98f6dd573aeb8ed79fa4a689603c5d1aa
contracts/LsdRateOracle.sol	90411f81c11d754aab3c554851047f5da95ae52e
contracts/Vault.sol	92d2d04c878f76d949d9b738ab462d3a89db94ce
contracts/xF33dSender.sol	26f59339c7a6aec9b0aaa277c729b04dfcbbac4d

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).



Dependency / Import Path	Count
@openzeppelin/contracts/access/Ownable2Step.sol	1
@openzeppelin/contracts/governance/utils/IVotes.sol	1
@openzeppelin/contracts/interfaces/IERC4906.sol	1
@openzeppelin/contracts/security/ReentrancyGuard.sol	2
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	1
@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol	1
@openzeppelin/contracts/utils/math/Math.sol	1
contracts/interfaces/IBribe.sol	1
contracts/interfaces/IERC20.sol	1
contracts/interfaces/IGauge.sol	1
contracts/interfaces/IVault.sol	1
contracts/interfaces/IVoter.sol	1
contracts/interfaces/IVotingEscrow.sol	1
contracts/libraries/Math.sol	1

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way.

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable.

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
8	0	12	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
205	3

External	Internal	Private	Pure	View
189	127	0	0	108

StateVariables

Total	🌐Public
55	44

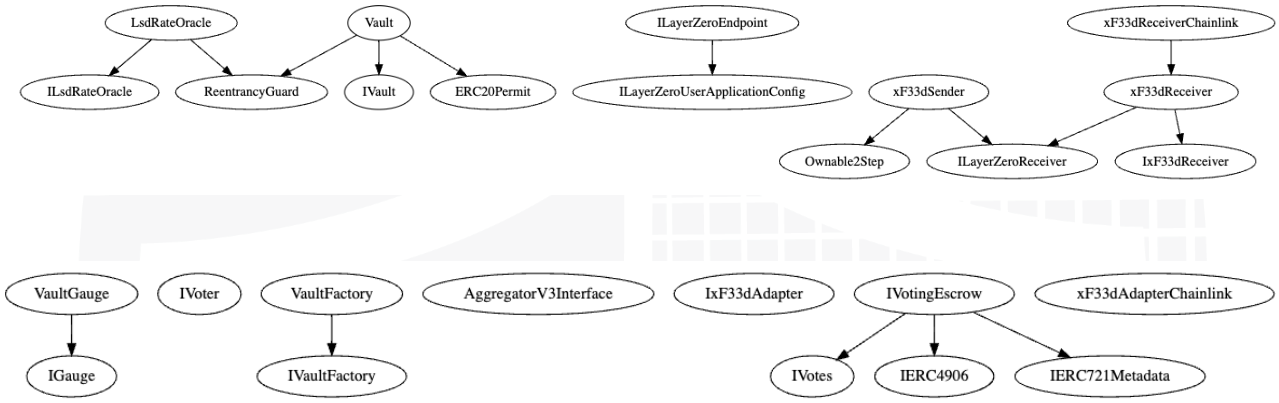
Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts

`^0.8.0`
`>=0.5.0`
`0.8.13`
`^0.8.13`
	----	Yes	Yes	----

Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit vulnerability and the impact of that event on the organization or system. The risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
    a. Reviewing the specifications, sources, and instructions provided to SolidProof to ensure we understand the size, scope, and functionality of the smart contract.
    b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
    c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.

2. Testing and automated analysis that includes the following:
    a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
    b. Symbolic execution, which is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.

4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

Overall Security

Medium or higher issues

Upgradeability

Contract is not an upgradeable	✅ Deployer cannot update the contract with new functionalities
Description	The contract is not an upgradeable contract. The deployer is not able to change or add any functionalities to the contract after deploying.
Comment	N/A

Ownership

The ownership is not renounced	❌ **The owner is not renounce**
Description	The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including: • Centralizations • The owner has significant control over contract's operations
Comment	N/A

Note - If the contract is not deployed then we would consider the ownership to be not renounced. Moreover, if there are no ownership functionalities then the ownership is automatically considered renounced.

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refer to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who has the ability to add new tokens to the network's total supply.

Contract owner cannot mint new tokens	✅ **The owner cannot mint new tokens**
Description	The owner is not able to mint new tokens once the contract is deployed.
Comment	N/A

Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens	✅ **The owner cannot burn tokens**
Description	The owner is not able burn tokens without any allowances.
Comment	N/A

Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses	✅ **The owner cannot blacklist addresses**
Description	The owner is not able blacklist addresses to lock funds.
Comment	N/A

Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner cannot set fees more than 25%	✅ **The owner cannot levy unfair taxes**
Description	The owner is not able to set the fees above 25%
Comment	N/A

Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When tokens or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Owner cannot lock the contract	✅ **The owner cannot lock the contract**
Description	The owner is not able to lock the contract by any functions or updating any variables.
Comment	N/A

Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project, there are authorities that have access to the following functions:

File	Privileges
1. Vault.sol	❖ onlyFactory – Set Gauge Address
2. xF33dSender.sol	❖ onlyOwner – Set Remote src address – Set protected feeds – Set Lz Endpoint address

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g. Gnosis Safe
- Use of a timelock at least with a latency of e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement
- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.

Audit Results

#1 | Logical Error

File	Severity	Location	Status
xF33dReceiver	Medium	L60	Fixed

Description - The xF33dAdapterChainlink is encoding following type: uint80, '*INT256*', uint256, uint256, uint80 and it is being decoded as uint80, '*UINT256*', uint256, uint256, uint80. When the chainlink getLatestData price is negative, the rate will result in an underflow.

Remediation - Make sure to decode with the same data type to avoid underflow

#2 | Missing Timelock

File	Severity	Location	Status
VaultGauge	Low	L491	ACK

Description - The contract misses a timelock in the withdraw function. This means that the claim function can be called recursively and the funds will be available for withdrawal right after the deposit

Remediation - We recommend putting a timelock so that the withdraw function cannot be called by an external contract recursively and only legitimate users will be able to withdraw funds.

Alleviation - This bug doesn't affect the business logic of the contract

#3 | Missing Zero Address Validation

File	Severity	Location	Status
VaultGauge	Low	L87—91	Fixed

Description - Make sure to validate that the address passed in the function parameters is "non-zero".

#4 | Missing Zero Address Validation

File	Severity	Location	Status
xF33dReceiver	Low	L33	Fixed

Description - Make sure to validate that the address passed in the function parameters is “non-zero”.

#5 | Missing Zero Address Validation

File	Severity	Location	Status
xF33dSender	Low	L88, 90, 113	Fixed

Description - Make sure to validate that the address passed in the function parameters is “non-zero”.

#6 | ChainID Check

File	Severity	Location	Status
xF33dSender	Low	L87	Fixed

Description - We recommend using the enumerable set to check for the valid chainIDs

#7 | Missng “msg.value” check

File	Severity	Location	Status
xF33dSender	Low	L66, 118	Fixed

Description - The contract doesn’t have any checks to verify whether the msg.value is zero or not. This may result in users calling this function with almost no ether.

Remediation - Check that the “msg.value” is not zero.

#8 | Missing Zero Address Validation

File	Severity	Location	Status
xF33dReceiver	Low	L34, 35, 37	Fixed

Description - Make sure to validate that the address passed in the function parameters is “non-zero”.

#9 | Missing Zero Address Validation

File	Severity	Location	Status
LsdRateOracle	Low	L14, 15	Fixed

Description - Make sure to validate that the address passed in the function parameters is “non-zero”.

#10 | Missing Zero Address Validation

File	Severity	Location	Status
Vault	Low	L48, 49	Fixed

Description - Make sure to validate that the address passed in the function parameters is “non-zero”.

#11 | Missing Logical Check

File	Severity	Location	Status
Vault	Low	L90	Fixed

Description - Make sure to check that the “totalLsd” value must be greater than or equal to “userLsd”

#12 | Hardcoded Address

File	Severity	Location	Status
Vault	Low	L86	Fixed

Description - The function uses a Hardcoded address as access control which is not recommended in any case because this address will always be in the control of this function even if it is hacked or the responsible person is removed from the project.

Remediation - We recommend storing the address in a variable and checking it from there.

#13 | Too Many Digits

File	Severity	Location	Status
Vault	Informational	L103, 112	Fixed

Description - Too many digits are being used for comparison. It is recommended to use the ether notation and if this is not possible the n the underscore notation (1_000_000) is also suffice.

#14 | Syntax Error

File	Severity	Location	Status
LsdRateOracle	Informational	L23	Fixed

Description - “External” keyword is used before the “view” keyword.

#15 | Business Logic Risk

File	Severity	Location	Status
LsdRateOracle	Informational	L19	Fixed

Description - The governor address is not able to remove addresses from “_rateManager” mapping. Beware while using this function.

#16 | Missng Error Message

File	Severity	Location	Status
xF33dSender	Informational	L147, 152	Fixed

Description - We recommend returning error messages in the case where the “require” check fails.

#17 | NatSpec documentation missing

File	Severity	Location	Status
All	Informational	—	ACK

Description - If you started to comment on your code, also comment on all other functions, variables etc.

#18 | Missng Error Message

File	Severity	Location	Status
xF33dReceiver	Informational	L46, 51	Fixed

Description - We recommend returning error messages in the case where the “require” check fails.

#19 | Missng Error Message

File	Severity	Location	Status
VaultGauge	Informational	L105, 526, 527, 542, 546, 560, 567, 570, 577, 584	Fixed

Description - We recommend returning error messages in the case where the “require” check fails.

#20 | Contract doesn’t import npm packages from source (like OpenZeppelin etc.)

File	Severity	Location	Status
All	Informational	N/A	ACK

Description - We recommend importing all packages from npm directly without flattening the contract. Functions could be modified or can be susceptible to vulnerabilities.

Note for the Project Team and Investors - Some interfaces used in the contracts’ logic like “IVoter”, “IVotingEscrow” was not the part of the audit scope and we cannot comment on their security.

Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.

Solid
Proofed

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY