SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY

DYOR Labs

AUDIT

SECURITY ASSESSMENT

29. October, 2024

FOR

SolidProof_io

@solidproof_io

Introduction

SolidProof.io is a brand of the officially registered company Future Visions Deutschland. We're mainly focused on Blockchain Security, such as Smart Contract Audits and KYC verification for project teams.
Solidproof.io assesses potential security issues in the smart contracts implementations, reviews for potential inconsistencies between the code base and the whitepaper/documentation, and provides suggestions for improvement.

Disclaimer

SolidProof.io reports are not, nor should they be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should they be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io does not cover testing or auditing the integration with external contracts 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 analysed, 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 analyse.

Project Overview

Summary

Project Name	DYOR Labs
Website	https://www.dyorlabs.com/
About the project	DYOR brings everything you need to manage, grow, and scale your crypto project into one seamless platform. Whether you're launching a new token, managing a thriving community, or building long-term strategies, we've got you covered. Our dashboard integrates essential tools for transparency, marketing, analytics, and security, all in one user-friendly interface.
Chain	TBA
Language	Solidity
Codebase Link	Provided as files.
Commit	N/A
Unit Tests	Provided

Social Medias

Telegram	https://t.me/dyor_labs
Twitter	https://x.com/dyorlabs_
Facebook	N/A
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	N/A
Discord	https://discord.com/invite/PWxkhnAd8A
Youtube	N/A
TikTok	N/A
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Changelog
v1.0	18. October 2024	• Layout Project • Automated- /Manual-Security Testing • Summary
v1.1	29. October 2024	• Reaudit

Note - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project that includes malicious outside manipulation of the contract's functions. 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 we did not functionally test it. This includes internal calculations in the formulae used in the contract.

File Overview

The Team 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-dyor/vault/storage/VaultStorage.sol	058262fc555fcd9059dac388313c6ea45c6656e1
contracts-dyor/vault/signatures/interfaces/IERC1271.sol	1ba4fea22247408f76cd649a35d51e66ebf6a42a
contracts-dyor/vault/signatures/interfaces/IEIP712.sol	b087ae2cc636278d55ff4cc59c080c0292e4e86e
contracts-dyor/vault/signatures/EIP712.sol	eceb39c6fb1973a1a2e4f9a58fa937c1ed3a3698
contracts-dyor/vault/signatures/libraries/SignatureVerification.sol	287065776441392f5b01094471e8cb49dd80ab49
contracts-dyor/vault/signatures/libraries/WithdrawalHash.sol	a6e9a6ceb43ddd595788e765a11f15619d006372
contracts-dyor/vault/signatures/WithdrawalHash.sol	c80c3c3b520ae4539db488d88f180fd16fe262b3
contracts-dyor/vault/upgradeInitializers/VaultInit.sol	aaf9af62ecb0f554840968e2f737ad3cc91b8740
contracts-dyor/vault/facets/VaultFacet.sol	8c08ed472285421cd5446dc8f4677892413f77d1
contracts-dyor/staking/signatures/ClaimHash.sol	51a700e435f114e76538a86b752f9fd106a2ea00
contracts-dyor/staking/interfaces/IStakingFacet.sol	8ba97fff6916a3880e084e5317643ab51f2ce7f0
contracts-dyor/cto/facets/CTOFacet.sol	a0c280049da51d4ca8688217167e5563e737e943
contracts-auth/Role.sol	66965c218aecfb75512d19c2682b0e1517129aac
contracts-auth/Authorizable.sol	5175f69c459d8fbb96beb48ac00d0b2ff780c7b8
contracts-dyor/staking/storage/StakingStorage.sol	8acaf1695bfc60a5286459c3dbdabd4e6077549a
contracts-signatures/interfaces/IERC1271.sol	1ba4fea22247408f76cd649a35d51e66ebf6a42a
contracts-signatures/interfaces/IEIP712.sol	b087ae2cc636278d55ff4cc59c080c0292e4e86e

contracts-dyor/token/storage/DYORStorage.sol	310964857a6e72555ea779b0a16c6fb6711a0ea6
contracts-dyor/token/inits/DYORInit.sol	2429c8c67946c7dd4d723fd297923eb493893ddb
contracts-common/interfaces/IWETH9.sol	2a870298d7642a51161b216bd995da0099110820
contracts-common/interfaces/IERC1155Receiver.sol	a2844517fd0730a41f7aa54ebae53a59cbc7cc20
contracts-signatures/EIP712.sol	859b29fadcf044610e532ac97f39dc48e4901169
contracts-dyor/staking/inits/StakingInit.sol	b777915d1dae6facfa7364d30caf88e67cd137a5
contracts-signatures/libraries/SignatureVerification.sol	287065776441392f5b01094471e8cb49dd80ab49
contracts-dyor/staking/facets/StakingFacet.sol	17f2b56b6df249b76f8635fca9ec89ca9c270a9f
contracts-common/contracts/ReentrancyGuardTransient.sol	bb93252d49f2419e0d583811dd8ee4907b016061
contracts-common/libraries/Constants.sol	624d327ab3243e3244e5c144cb06e09874263754
contracts-common/libraries/Arrays.sol	bece17d54b081f5d501910d15c8fea04920eb6cc
contracts-common/libraries/Methods.sol	91d05dee9559096a5746c8d136807858410d3f67
contracts-common/libraries/BlockTimestamp.sol	9c773261b9973e94e65cc653709ec49a651b31e7
contracts-common/libraries/BlockNumber.sol	58cbefa62502de1837391142d6d4d1384c299f9c
contracts-common/libraries/Strings.sol	6da9ad08fea9060cc416523c9ff8a91541d2a56e
contracts-common/libraries/ChainId.sol	e5b74e461c1ce654a621c7416fdfbb8daea8d7b8
contracts-dyor/cto/inits/CTOInit.sol	0ded3b38bf8681bb025bc68336a77a3f534d826f
contracts-auth/interfaces/IAuthorizationProvider.sol	14640b2a34a00b51902c4c3e25e9e67d7f00b465
contracts-auth/providers/RoleAuthorizableProvider.sol	7f047e9be50df37f0a3fd423641887960e1ced47
contracts-auth/interfaces/IAuthorizableProvider.sol	5c55785a657b1797fe297a1d61b44cc092bf56a3

contracts-auth/interfaces/IAuthorizable.sol	4356007271da724fd157e67c02781949fa6bd31b
contracts-internal/base/ForkBase.sol	d567ff40cdaf4240eda4cf3926e304dc6ac041f7
contracts-internal/base/AccountBase.sol	ff6bc0861ddf1731136d0569af96aec40e152cf6
contracts-internal/eip2535/IFacet.sol	75948d79b7eadea2836556bb03412eec34944802
contracts-internal/eip2535/Facet.sol	f16b94c10d9a247f5818ebdc353230519a838eb7
contracts-internal/base/ChainBase.sol	ad9dc2138c225f6f0c6060232d9dbb0da0a755b1
contracts-internal/eip2535/DiamondBase.sol	9646a7cb267d3715bd3e75d35c3b53bbd7b28ce5
contracts-internal/eip2535/IInitializer.sol	e1e9ecb0e97f7e1e502202e24ce78a6af481279d
contracts-exchange/v2/core/interfaces/IUniswapV2Callee.sol	1a28049374306d4e5b0e93eae93f50f2271e27c8
contracts-exchange/v2/core/interfaces/IUniswapV2Pair.sol	a2dee749d02223289b9a3beb344ca94e81332fac
contracts-exchange/v2/shared/interfaces/IERC20.sol	c0bcaf34e45b9f72ecc792af1703f7c1c9da7d42
contracts-exchange/v2/shared/libraries/FullMath.sol	335f967e9f96c4e0ff7af0f24008ca0fe72198bb
contracts-exchange/v2/shared/libraries/TransferHelper.sol	0cae72afadac4d37238307cf567009649de61643
contracts-exchange/v2/shared/libraries/Babylonian.sol	3465603982320184e168e9b3028f4e7c027f8db2
contracts-exchange/v2/shared/libraries/FixedPoint.sol	1b16a021802730439bad27a447839790e85b5291
contracts-exchange/v2/shared/libraries/BitMath.sol	21cf242f6cd6395ccb70708429d97d9379b1b590
contracts-exchange/v2/core/UniswapV2Pair.sol	9d6358c1e1a15d95e636b3afcf03ea15893c6b2f
contracts-exchange/v2/core/libraries/UQ112x112.sol	8f64b54346704c1a03d40a677d68b36bf1c47441
contracts-exchange/v2/core/libraries/Math.sol	54a307f0f11b09bf0e87b6d4b48eb4983acea59a
contracts-exchange/v2/core/factory/facets/UniswapV2FactoryFacet.sol	e14b3f320a489de0f4da4a4c027688516fd6057d

contracts-exchange/v2/core/factory/interfaces/IUniswapV2Factory.sol	cef315e72151cc230d5573afe3a0987e63c69591
contracts-exchange/v2/periphery/interfaces/IWETH.sol	a950d156dfdb6a6c9737090cae33a07eb8bd085c
contracts-exchange/v2/periphery/interfaces/IUniswapV2Router01.sol	04cfcab4d144adb62bdb8e597d902dbdfee09a46
contracts-exchange/v2/periphery/interfaces/IUniswapV2Router02.sol	ccb239755b12c775cbb4665fb870803e32354495
contracts-exchange/v2/core/factory/storage/UniswapV2FactoryStorage.sol	f6bf8e171a221945321129a7c99c2c0bd3b3608b
contracts-exchange/v2/core/factory/upgradeInitializers/UniswapV2FactoryInit.sol	eb23d9e1090d10a8c20d5b146b486cd201353995
contracts-exchange/v2/periphery/upgradeInitializers/UniswapV2RouterInit.sol	a426a6e49e1ec5f3726f6997867f6056c777f11b
contracts-exchange/v2/periphery/facets/UpdatePairFacet.sol	f0ef763015c0a98423727e86834538122e5f9f0f
contracts-exchange/v2/periphery/facets/CoreRouterFacet.sol	c3b3eb00d1de44730c722d0344dcefaac9c35e1e
contracts-exchange/v2/periphery/libraries/UniswapV2LiquidityMathLibrary.sol	1e995d94cac684e061a00801afca88d7bee98200
contracts-exchange/v2/periphery/libraries/UniswapV2Library.sol	31e27585c05eb27d80e17389752c7d4569e36606
contracts-exchange/v2/periphery/libraries/UniswapV2OracleLibrary.sol	28ef4318f638a9db34412f021ac373750adeed47

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) indicate 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
@auth/Authorizable.sol	4
@auth/Role.sol	1
@auth/interfaces/IAuthorizable.sol	2
@auth/interfaces/IAuthorizableProvider.sol	1
@auth/interfaces/IAuthorizationProvider.sol	2
@common/contracts/ReentrancyGuardTransient.sol	3
@common/libraries/Arrays.sol	1
@common/libraries/BlockTimestamp.sol	2
@common/libraries/Methods.sol	1
@common/libraries/Strings.sol	3
@internal/base/ChainBase.sol	1
@internal/base/ForkBase.sol	1
@internal/eip2535/DiamondBase.sol	1
@internal/eip2535/Facet.sol	6
@internal/eip2535/IFacet.sol	1
@internal/eip2535/IInitializer.sol	7
@openzeppelin/contracts/interfaces/IERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	2
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	2
@openzeppelin/contracts/utils/Strings.sol	2
@openzeppelin/contracts/utils/introspection/ERC165Checker.sol	1
@openzeppelin/contracts/utils/structs/EnumerableSet.sol	1
@openzeppelin/contracts/vendor/arbitrum/IArbSys.sol	1
@permit2/interfaces/IPermit2.sol	1
@signatures/EIP712.sol	2
@signatures/interfaces/IEIP712.sol	2

@signatures/interfaces/IERC1271.sol	2
@signatures/libraries/SignatureVerification.sol	2
@solidstate/access/ownable/OwnableStorage.sol	1
@solidstate/data/EnumerableSet.sol	4
@solidstate/interfaces/IERC165.sol	2
@solidstate/interfaces/IERC20.sol	2
@solidstate/introspection/ERC165/base/ERC165Base.sol	4
@solidstate/proxy/diamond/ISolidStateDiamond.sol	1
@solidstate/proxy/diamond/SolidStateDiamond.sol	1
@solidstate/proxy/diamond/readable/IDiamondReadable.sol	1
@solidstate/proxy/diamond/writable/IDiamondWritableInternal.sol	1
@solidstate/token/ERC20/SolidStateERC20.sol	1
@solidstate/token/ERC20/metadata/ERC20MetadataStorage.sol	1
@solidstate/token/ERC20/metadata/IERC20Metadata.sol	1
@solidstate/token/ERC20/permit/IERC20Permit.sol	1
@vulcan/_internal/Utils.sol	1
forge-std/Script.sol	1
forge-std/StdCheats.sol	1
forge-std/Test.sol	1
forge-std/Vm.sol	4

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

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. Review the specifications, sources, and instructions provided to SolidProof to ensure we understand the smart contract's size, scope, and functionality.
    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 is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., 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

Upgradeability

Contract is an upgradeable	❌ Deployer can update the contract with new functionalities
Description	The deployer can replace the old contract with a new one with new features. Be aware of this, because the owner can add new features that may have a negative impact on your investments.
Example	We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.
Comment	The contract employs the Diamond pattern (EIP-2535), which allows for modular upgrades by adding or modifying facets to introduce new functionality. While this design offers flexibility and scalability, it also presents potential risks if not carefully managed. Unauthorized or unverified upgrades could lead to unintended changes in contract behavior, which may disrupt functionality or affect user balances. To mitigate these risks, it's essential to implement strong governance processes. In scenarios where immutability is preferred for security, using non-upgradable contracts may be a more prudent approach.
Alleviation	The project owner has acknowledged and confirm it as a part of business logic.

Ownership

Contract ownership is not renounced	❌ The ownership is not renounced
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
Example	We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.
Comment	The project owner has acknowledged and confirm it as a part of business logic.

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 refers 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 can 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 without Allowance

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 to 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 to 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 contract's cost, 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.

Contract 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

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
21	30	19	5

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
185	16

External	Internal	Private	Pure	View
168	307	12	56	107

StateVariables

Total	🌐Public
119	24

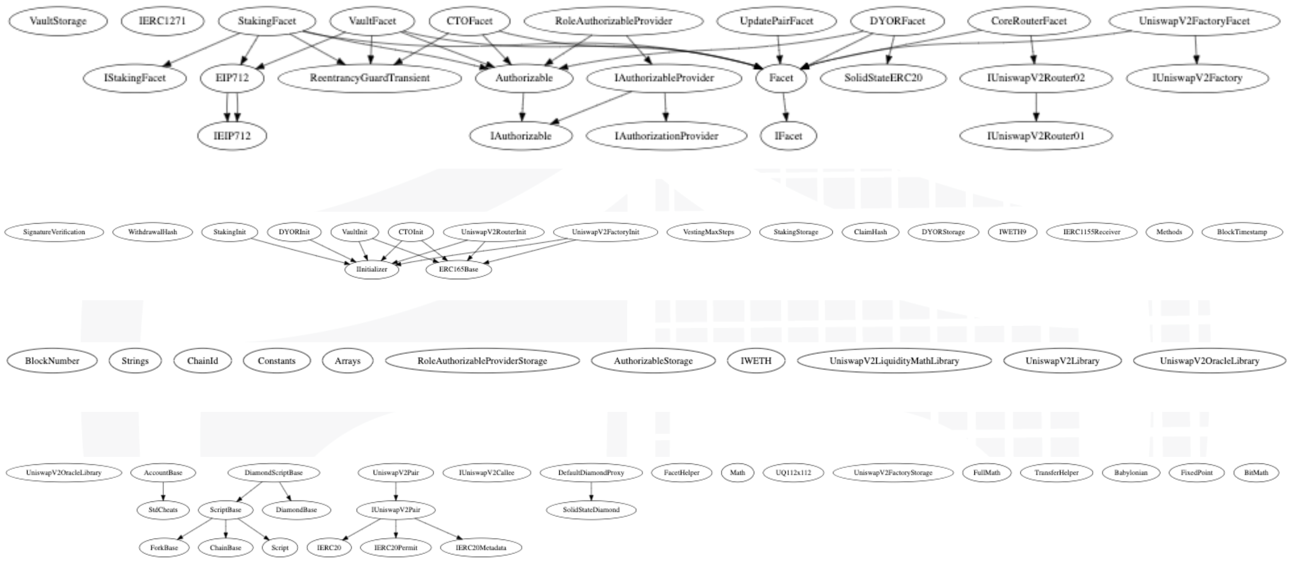
Capabilities

Solidity Versions observed	Transfers ETH	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.0` `^0.8.17`		Yes	yes (16 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
yes			yes	yes	yes → AssemblyCall:Name:create2 → NewContract:DefaultDiamondProxy → AssemblyCall:Name:create

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.



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 a single entity controls the contract or if certain participants have special permissions or abilities that others do not.

In the project, some authorities have access to the following functions:

File	Privileges
Authorizable.sol	• The deployer or authorized users can set the authorization provider in the contract.
CTOFacet.sol	• The deployer or authorized users can withdraw the USDC balance from the contract.
StakingFacet.sol	• The deployer or authorized users can deposit amount for the users. • The deployer or authorised users can withdraw tokens for the user. • The deployer or authorised addresses can deposit USDC tokens to the contract. • The deployer or authorised addresses can update the vesting stage. • The deployer or authorised addresses can initialize the staking only once.
VaultFacet.sol	• The deployer or authorized addresses can send the deposited USDC tokens to the multi-sig wallet. • The deployer or authorized addresses can decrease the token amount on a particular account.

Recommendations

To avoid potential hacking risks, the client should 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 can no longer modify any state variables of the contract. Make sure to set up everything before renouncing.

Audit Results

Critical issues

No critical issues

High issues

#1 I Manipulation of funds.

File	Severity	Location	Status
VaultFacet.sol	High	L52-54	ACK

Description - The updateVaultAccountBalance function allows the reduction of an account's balance without verifying if the update is legitimate, which could lead to unauthorized loss of funds. Also, This is a security risk. An authorized actor could reduce the balance arbitrarily without proper validation, potentially leading to theft or unintentional balance reductions.

Remediation - Add a validation mechanism such as requiring a signature or an additional check to ensure that only valid balance updates occur.

Medium issues

No medium issues

Low issues

#1 I Floating pragma solidity version.

File	Severity	Location	Status
All	Low	–	ACK

Description - The contracts should be deployed with the same compiler version and flag that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions.

#2 I Missing Visibility.

File	Severity	Location	Status
All	Low	–	ACK

Description - It is recommended to add the ‘public’, ‘private’, ‘internal’ visibility during the initialisation of a state variable or a mapping in the contract.

Informational issues

#1 I NatSpec documentation missing

File	Severity	Location	Status
All	Informational	–	ACK

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

#2 I Contract doesn’t import 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.

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