SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Start / Vault

AUDIT

SECURITY ASSESSMENT

31. July, 2023

FOR

SOLIDProof

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	USDFI - Start / Vault
Website	https://usdfi.com/
About the project	USDFI is crypto's first and only Universal Banking Protocol and DeFi's multichain native banking layer.
Chain	BSC
Language	Solidity
Codebase Link	https://github.com/AureumVictoria/Audit2
Commit	https://github.com/AureumVictoria/Audit2/commit/5f28f6d662eecf136dc11c31c6a80d0f117b1cf1
Unit Tests	Not Provided

Social Medias

|	
Telegram	https://t.me/USDFI
Twitter	https://twitter.com/stable_usdfi
Facebook	N/A
Instagram	https://www.instagram.com/stable.usdfi/
Github	https://github.com/USDFI
Reddit	N/A
Medium	https://medium.com/@usdfi
Discord	https://discord.gg/MjvpF8UwB4
Youtube	https://www.youtube.com/@usdfi
TikTok	N/A
LinkedIn	N/A

Comments

- Referrals contract was not provided to Solidproof
- GasPrice contract was not provided to Solidproof
- RewardPool was referenced to https://bscscan.com/address/0x56996C3686E131A73E512d35308f348f987Bc0D5#code

Audit Summary

Version	Delivery Date	Changelog
v1.0	26. July 2023	• Layout Project • Automated- /Manual-Security Testing • Summary
	31. July 2023	• Reaudit

Note - This Audit report consists of a security analysis of the smart contract in the GitHub repo above. This analysis did not include functional testing (or unit testing) of the contract's logic.
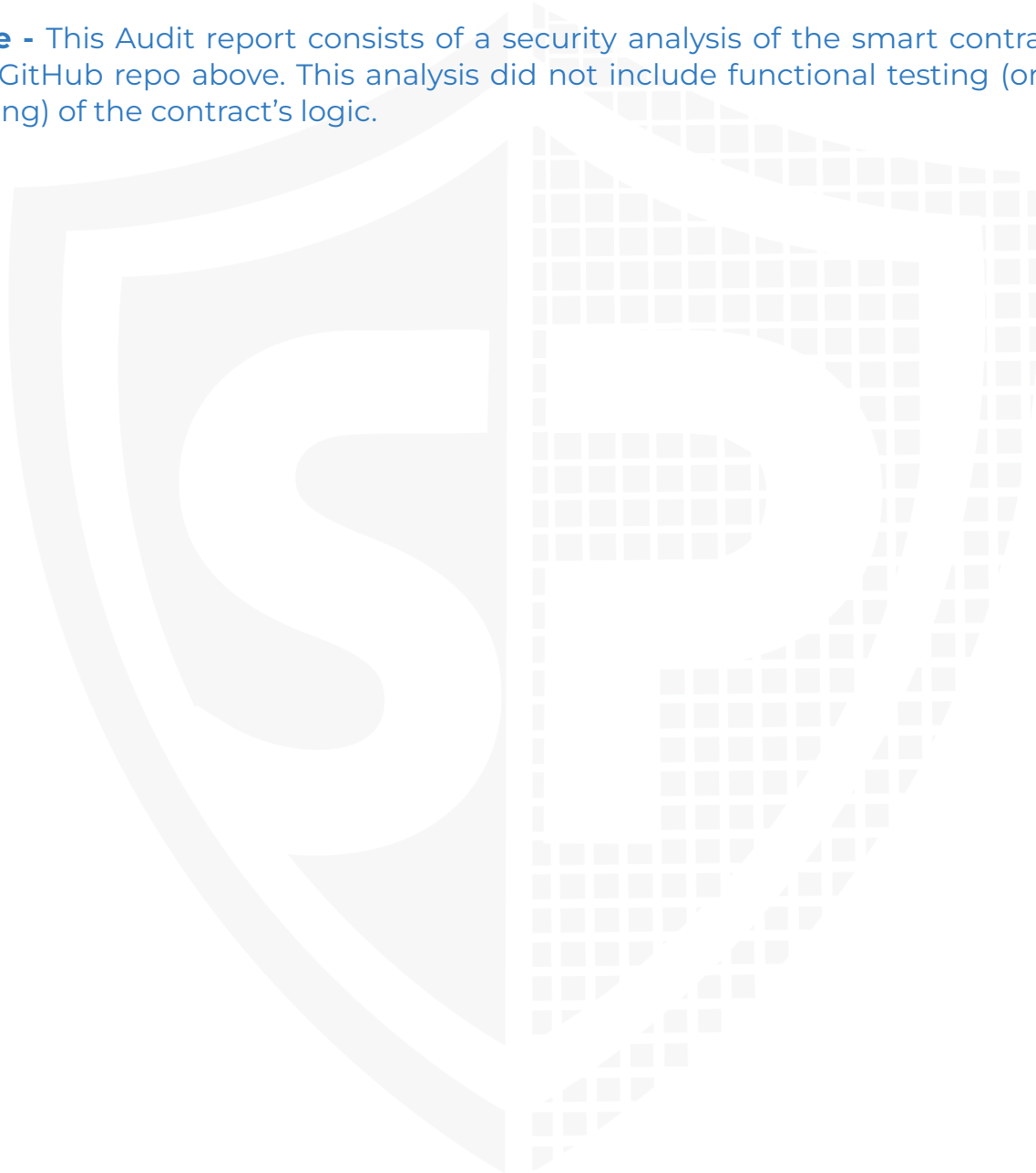
File Overview

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/IERC20Upgradeable.sol	05598e8316056416fa82c26675d7df1f82422c56
contracts/AddressUpgradeable.sol	ab01b1cde3da2f29e3dede5163ba19e476a13421
contracts/SafeERC20Upgradeable.sol	1bd692e4eecbf2784af7afef3455bf0e42105d7a
contracts/IReferrals.sol	e35371fefbbee18858f66f2d93e96bf5ea028b87
contracts/IERC20MetadataUpgradeable.sol	881c4eaef26a4e82a10a027325f83576e0b85787
contracts/start.sol	c052dc8434df524691ee1057225f4a645c2c5781
contracts/IERC20PermitUpgradeable.sol	0badba90cc792b9b8cc0b0bb535728fd169abbaa
contracts/IStrategy.sol	e1c9c909863c8597ce5dd581e6b2189b91dfbea2
contracts/Vault.sol	e50cb2d8b6a7a9b0c7827602b11f2f04319e2278
contracts/ReentrancyGuardUpgradeable.sol	c1626db251909af8a43caf93c223503aacdf54eb
contracts/ContextUpgradeable.sol	2221f6ca533c85b91662d22c549443bdd8da3d24
contracts/OwnableUpgradeable.sol	5435281e6fbcef01292dfe6c70cd449cff192f0e
contracts/ERC20Upgradeable.sol	4b5068a46fef382045ae9ed45a20908c3e7555a7
contracts/Initializable.sol	8600375453e118256c73dd1f8f3e00a9dabb268c

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).

- AddressUpgradeable.sol
- ContextUpgradeable.sol
- ERC20Upgradeable.sol
- IERC20MetadataUpgradeable.sol
- IERC20PermitUpgradeable.sol
- IERC20Upgradeable.sol
- Initializable.sol
- IReferrals.sol
- IStrategy.sol
- OwnableUpgradeable.sol
- ReentrancyGuardUpgradeable.sol
- SafeERC20Upgradeable.sol

Note for Investors: We only Audited the start/vault contract for the USDFI Project. However, If the project has other contracts (for example, a Presale contract etc), and they were not provided to us in the audit scope, then 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. 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
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	Since the contract was not deployed we cannot make sure that this contract is an upgradeable or not. However, the contract is using upgradeable contracts in it.

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
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	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 can set fees greater than 25%	❌ The owner able to charge unfair fees/taxes
Description	For example, a decentralized exchange (DEX) smart contract may charge a fee for each trade executed on the platform. This fee can be set by the owner of the contract and may be a percentage of the trade value or a flat fee. In other cases, the owner of the smart contract may set fees for accessing or using certain features of the contract. For instance, a subscription-based service smart contract may charge a monthly or yearly fee for access to premium features. It's important to note that the fees set by the owner of a smart contract may not be the same as the gas fees required to execute the contract on the blockchain. Gas fees are generally set by the network and vary based on factors such as network congestion and the complexity of the transaction. The fees set by the contract owner, on the other hand, are independent of gas fees and are typically charged in addition to gas fees. Overall, fees set by the owner of a smart contract can provide an additional source of revenue for the contract's owner and can help to ensure the sustainability of the contract over time.
Example	Our assumption is that the owner can adjust the transfer, development, and marketing fees up to 100%. If the transfer fee is set to 100%, it implies that the full amount of tokens you intend to send will be sent to the address specified as the recipient in the contract. This implies that the recipient will never have the intended amount of tokens in their wallet as it has all been used up in paying for the transfer fee.
Comment	Withdrawal Fee can be set to max 0.5% but the compundPercent can be set to 100%
Alleviation	The user will get the compound fee.

File, Line/s: Start.sol, L2350, L2343

Codebase:

```
2347          // adjust withdrawal fee
              ftrace | funcSig
2348          function setWithdrawalFee(uint256 _fee↑) public onlyOwner {
2349              require(_fee↑ <= WITHDRAWAL_FEE_CAP, "!cap");
2350              withdrawalFee = _fee↑;
2351              emit SetWithdrawalFee(_fee↑);
2352          }
```

```
2030              WITHDRAWAL_FEE_CAP = 50;
2031              WITHDRAWAL_MAX = 10000;
```

```
2340          // adjust compound percent
              ftrace | funcSig
2341          function setCompoundPercent(uint256 _percent↑) public onlyOwner {
2342              require(_percent↑ <= 100, "!cap");
2343              compoundPercent = _percent↑;
2344              emit SetCompoundPercent(_percent↑);
2345          }
```

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	Owner can pause the deposit/harvest function only. Withdraw is still possible after pausing.

File, Line/s: Start.sol, L2260 - L2272
Codebase:

```
2260        function pause() public onlyOwner {
2261            _pause();
2262
2263            _removeAllowances();
2264        }
2265
        ftrace | funcSig
2266        function unpause() external onlyOwner {
2267            _unpause();
2268
2269            _giveAllowances();
2270
2271            deposit();
2272        }
2273
```

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
4	8	16	8

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
146	5

External	Internal	Private	Pure	View
102	268	8	24	91

StateVariables

Total	🌐Public
70	43

Capabilities

Solidity Versions observed	✏️ Experimental Features	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
^0.8.0 ^0.8.1 >=0.6.0 <0.9.0 ^0.8.2 >=0.8.0 <0.9.0 >=0.6.0 0.8.19	ABIEncoderV2	yes	yes (18 asm blocks)	—

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 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. Start.sol	❖ onlyOwner – Set – usdfiFeeRecipient – unirouter – Vault – withdrawalFee – compundPercent – nativeToLp0Path – outputToNativePath – shouldGasThrottle – harvestOnDeposit – isFastQuote – gammaProxy address – Give/Remove allowance in – Want address to rewardPool – Output addres to unirouter – Native to unirouter – Native to usdfiDex – lpToken0 to want address – lpToken1 to want address – Pause/unpause the contract – Panic call ❖ onlyVault – beforeDeposit – Withdraw – retireStrat – The vault is able to withdraw from the rewardPool and get out the tokens in the "want" address which is held by the StrategyThenaGamma contract. This tokens will be sent back to the vault contract.

File	Privileges	
2. Vault.sol	❖ onlyOwner - SetHarvestPause - inCaseTokensGetStuck - Update whitelist	21

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 | Missing Zero Address Validation

File	Severity	Location	Status
Start.sol	Low	L2368, L2362, L2355	ACK
Vault.sol	Low	L56, L57, L58,	ACK

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

Comment
- If the usdfiFeeRecipient is set to 0, the usdfiFeeAmount will be burnt
- If the unirouter is zero/dead address following functionalities will be reverted
  - addLiquidity
  - swapRewardsToNative

Alleviation
The team is aware of the zero/dead address issue. They are not going to check it because of the gas fees.

#2 | Disable Initializing

File	Severity	Location	Status
Start.sol	Low	L2004	ACK
Vault.sol	Low	L48	ACK

Description
– Ensure to disable the initialize function for updated versions in the contract after the first version is deployed. Otherwise, everybody can call the initialize function to reset the variables.

Comment
- The initialize function will be called automatically with the hardhat framework. Taking care of the function is recommended because everyone can call it later when the script is not set up properly.

Alleviation
- The disabling of the initializing should happen during the initializing because we are using the AdminUpgradeabilityProxy of the OpenZepplin library.

#3 | Unused state variables

File	Severity	Location	Status
start.sol	Low	L1683	ACK

Description
- Remove unused state variables

Alleviation
- It is a variable from the "FEE_SIZE" library. Acknowledged.

#4 | Uninitialized variable

File	Severity	Location	Status
start.sol	Low	L2163	ACK

Description
- Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

Alleviation
- We need the variable because of the function call "deposit" which expects a uint array with a length of 4. We don't need to initialize it because of the gas fees.

#5 | Local variables shadowing

File	Severity	Location	Status
Vault.sol	Low	L50 - L51	ACK

Description
- Rename the local variables that shadow another component

Alleviation
- Acknowledged

#6 | GenerateReward can be called directly

File	Severity	Location	Status
Vault.sol	Low	L277	ACK

Description
- The "generateReward" function will be called in the L233. This will only be called when the "harvestPause" is false. When it is paused, the claim function will not call it. Since there is a "generateRewards"

function, everybody can call it directly anyway. When it is supposed to be that the "generateReward" function can only be called when the harvest is not paused, it should be checked in the "_generateReward" also.

Alleviation
Once you have bots that create rewards, if there should be none, every 24h, one caller must take over the payment at the claim.

#7 | Old Compiler version

File	Severity	Location	Status
Start.sol	Low	L1672, L1734, etc.	Fixed

Description
- The contracts use outdated compiler versions, which are not recommended for deployment as they may be susceptible to known vulnerabilities.

#8 | NatSpec documentation missing

File	Severity	Location	Status
Start.sol	Informational	—	Low

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

Alleviation
Acknowledged.

#9 | Floating Pragma

File	Severity	Location	Status
Start.sol	Informational	L7, L153, L377, etc.	ACK

Description
- The current pragma Solidity directive is ">=0.6.0 <0.9.0". Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions

Alleviation

The floating pragmas are from the flattened files. You can see in L1921 that the certain pragma was set to "=0.8.21".

#10 | 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.

#11 | Solidity Types

File	Severity	Location	Status
Vault.sol	**Informational**	L76, L109, L137, L155, etc.	**Fixed**

Description

 – Use explicit types instead of general types. For example instead of using "uint" use "uint256"

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