QuillAudits

Audit Report
May, 2023

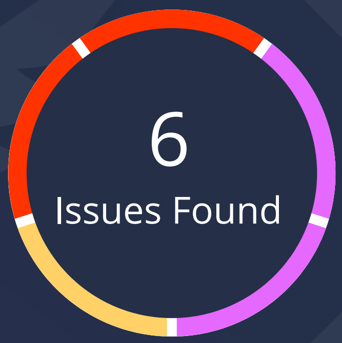
For

DappLooker

Table of Content

Executive Summary

Project Name	DappLooker
Overview	The PaymentVault contract serves as a vault to hold tokens deposited by users when subscriptions are made. The contract owner is able to withdraw the tokens deposited.
Timeline	21st March, 2023 to 14th April, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze DappLooker codebase for quality, security, and correctness. *https://github.com/dapplooker/contracts* **Branch:** main
Commit Hash	f7d796700d5d4ac1615608d78b12ef1a0faa5e6a
Contracts in Scope	PaymentVault.sol
Fixed in	*https://github.com/dapplooker/contracts* **Branch Name:** main **Commit Hash:** 186148120f563cc0c5884322958323f8e7ba5ea2



🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	**1**	0	0	0
Resolved Issues	**2**	**1**	0	**2**

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open
Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved
These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged
Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved
Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - PaymentVault.sol

High Severity Issues

1. Ownership and Privilege transfer

If renouncing ownership is not done appropriately, the current contract owner can renounce ownership, lose owner privileges, and lose access to the tokens sent to the contract forever. When ownership is renounced, functions like withdraw will be uncallable and any tokens gotten from transactions, or even mistaken deposits into the contract will be inaccessible.

Recommendation

Functions such as renounceOwnership and transferOwnership can be overridden or set up for 2-step verification to prevent mistaken privilege transfer or renouncing.

References - *Link 1* | *Link 2*

Status

Resolved

Auditor's Remark - The client has implemented SafeOwn's 2-step ownership transfers to mitigate the risk described above.

2. Token validity

The contract tries to restrict the usable tokens to only ERC20 tokens but their implementation under the hood can be quite different from what is expected from standard tokens. FeeOnTransfer tokens, and rebase tokens can drastically reduce the expected number of tokens deposited into the vault and the expected amount to be available for withdrawal.

Recommendation

Consider ways to handle tokens like these or have a list of approved tokens to be used in this project.

Status

Partially Resolved

Auditor's Remark - The client has implemented the SafeERC20 token wrapper to reduce the landscape of risk which custom ERC20 tokens introduce. To accommodate for future possible tokens to be added, no token list is used.

3. Frontrunning

Upon deployment, the expected contract owner would have to call initialize( ) else the owner characteristics and privileges do not get set. This is an issue for two reasons:

Scenario 1: Initialize is frontrun by a malicious actor
If anyone asides from the intended owner calls initialize, they automatically become the owner of this PaymentVault contract with the privileges required to call withdraw on any of the deposited tokens. Bots could be listening on the network for scenarios set up just like this and take advantage of the exploit opportunity.

Once the attacker is approved, they can set the owner to an auto-drain contract that immediately removes any deposits made into the vault. This could defeat the goal of having the contract act as a vault until needed.

Scenario 2: Initialize does not get called by anyone
This would make the owner variable set to address(0), and the withdraw function will be unusable since address(0) currently does not have a signer and cannot pass the onlyOwner check.

Recommendation

Ensure the deployer is aware of safe deployment practices. Also refer to OpenZeppelin's guide linked *here*.

Status

Resolved

Auditor's Remark - The contract has been made non-upgradeable to mitigate the risk of frontrunning.

Medium Severity Issues

4. Only owner can make withdrawals

The current logic of this contract allows only the owner to withdraw deposited tokens. This means User A can make a deposit but would never be able to withdraw their deposited tokens unless they were the deployers of the contract and/or the caller of the initialize function.

Recommendation

Update the logic to map tokens deposited to their owners and give withdrawal allocation to the token owners specifically or explicitly state the current functionality in the contract docs.

Status

Resolved

Dapplooker Team's Comment: We don't need to keep track which user have deposited and for what duration, we are simple keeping the transaction hash as proof that user have deposited and verifiying the increased amount for that particular token (which we know) by ERC20 token balance of function where we are looking for balance of our contract.
We are looking at contract, as an on chain wallet which will be have the token amount and as an owner we can withdraw to our wallet whenever required.

We didn't designed it like having a separate map to keep user wallets to transferred amount details.

Auditor's Remark - This tallies with the business logic as described by the client.

Low Severity Issues

No issues were found

Informational Issues

5. Event emission

The events emitted in the contract could be indexed to allow for easy searching for addresses that made deposits and the different tokens that were added in.

Recommendation

Consider including the token deposited as one of the parameters during Deposit event emission. Also, indexing would help in sorting through transactions.

Status

Resolved

Auditor's Remark - Events are now emitted for deposits and withdrawals.

6. Unlocked pragma

The solidity pragma version in this codebase is unlocked.

Recommendation

It is always advisable to use a specific Solidity version when deploying to production to reduce the surface of attacks with future releases which were not accounted for when the contracts went live.

Status

Resolved

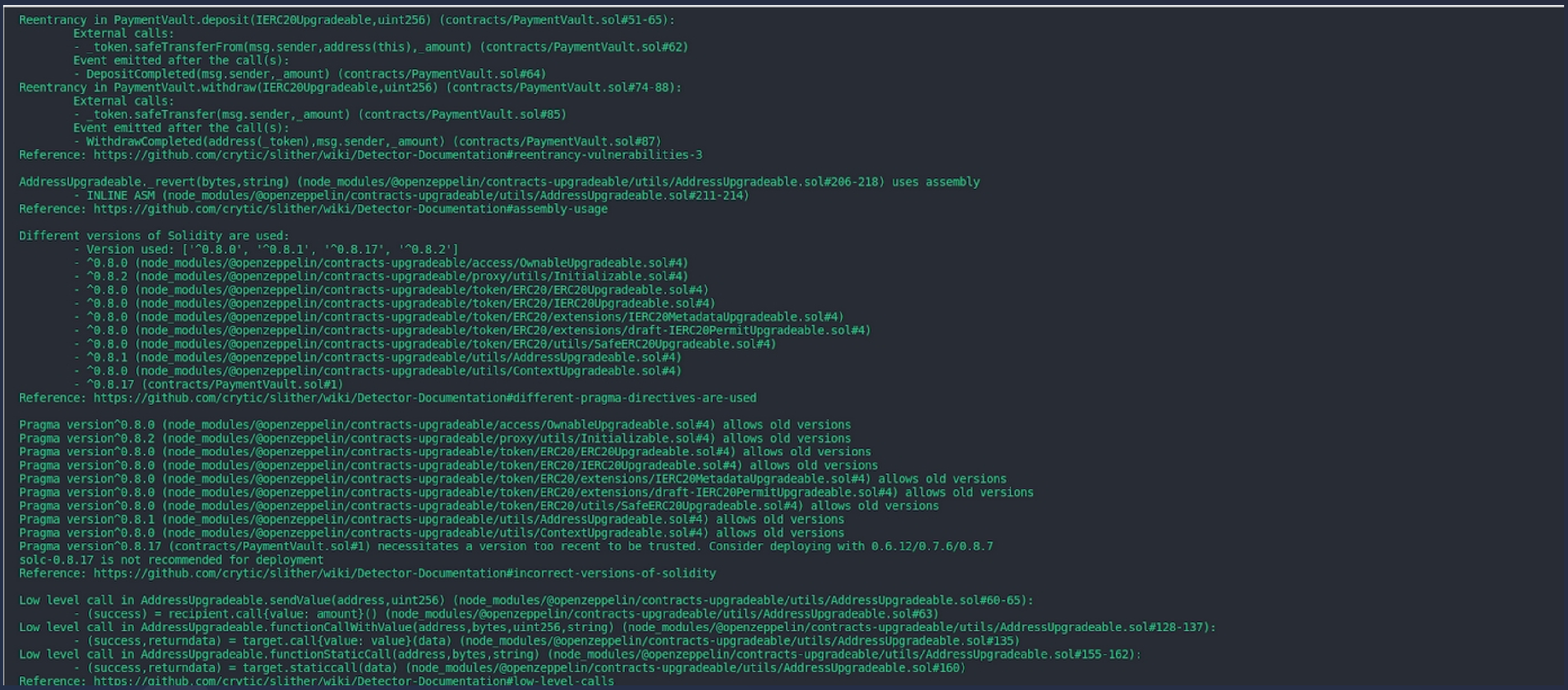
Auditor's Remark - Solidity version locked to 0.8.19

Functional Testing

	Test	Expectation	Actual	Verdict
✓	Deposit tokens	PASSED	PASSED	PASSED
✓	Withdraw tokens	PASSED	PASSED	PASSED
✓	Transfer ownership	PASSED	PASSED	PASSED
✓	Revoke ownership	PASSED	PASSED	PASSED
✓	Withdraw subscriptions by owner	PASSED	PASSED	PASSED

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.



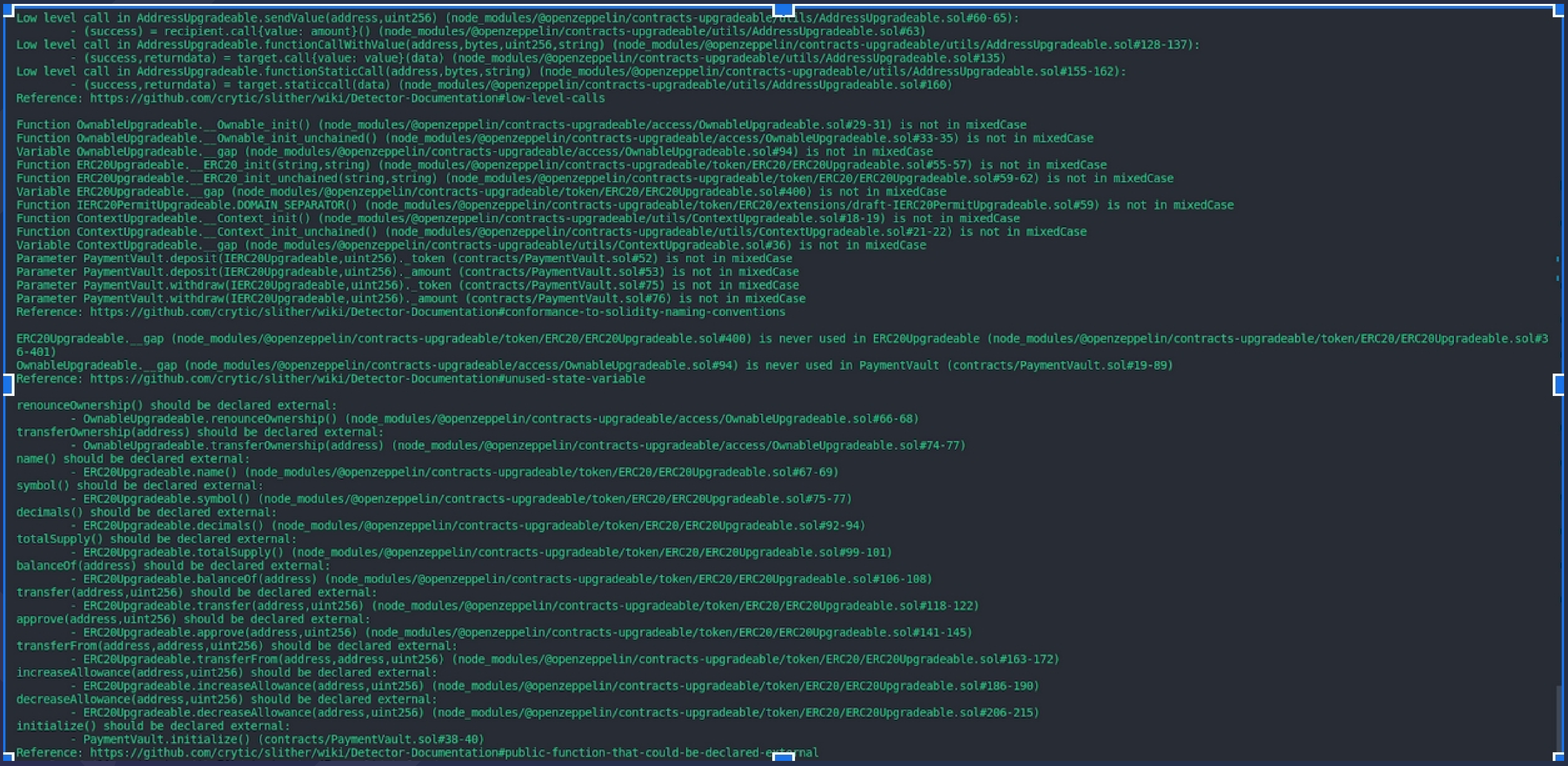
```
Reentrancy in PaymentVault.deposit(IERC20Upgradeable,uint256) (contracts/PaymentVault.sol#51-65):
        External calls:
        - _token.safeTransferFrom(msg.sender,address(this),_amount) (contracts/PaymentVault.sol#62)
        Event emitted after the call(s):
        - DepositCompleted(msg.sender,_amount) (contracts/PaymentVault.sol#64)
Reentrancy in PaymentVault.withdraw(IERC20Upgradeable,uint256) (contracts/PaymentVault.sol#74-88):
        External calls:
        - _token.safeTransfer(msg.sender,_amount) (contracts/PaymentVault.sol#85)
        Event emitted after the call(s):
        - WithdrawCompleted(address(_token),msg.sender,_amount) (contracts/PaymentVault.sol#87)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

AddressUpgradeable._revert(bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#206-218) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#211-214)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1', '^0.8.17', '^0.8.2']
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.17 (contracts/PaymentVault.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.17 (contracts/PaymentVault.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-137):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#155-162):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```



```
Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-137):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#155-162):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function OwnableUpgradeable.__Ownable_init() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#29-31) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#33-35) is not in mixedCase
Variable OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#94) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#55-57) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#59-62) is not in mixedCase
Variable ERC20Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#400) is not in mixedCase
Function IERC20PermitUpgradeable.DOMAIN_SEPARATOR() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol#59) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Parameter PaymentVault.deposit(IERC20Upgradeable,uint256)._token (contracts/PaymentVault.sol#52) is not in mixedCase
Parameter PaymentVault.deposit(IERC20Upgradeable,uint256)._amount (contracts/PaymentVault.sol#53) is not in mixedCase
Parameter PaymentVault.withdraw(IERC20Upgradeable,uint256)._token (contracts/PaymentVault.sol#75) is not in mixedCase
Parameter PaymentVault.withdraw(IERC20Upgradeable,uint256)._amount (contracts/PaymentVault.sol#76) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

ERC20Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#400) is never used in ERC20Upgradeable (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#3
6-401)
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#94) is never used in PaymentVault (contracts/PaymentVault.sol#19-89)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

renounceOwnership() should be declared external:
        - OwnableUpgradeable.renounceOwnership() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#66-68)
transferOwnership(address) should be declared external:
        - OwnableUpgradeable.transferOwnership(address) (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#74-77)
name() should be declared external:
        - ERC20Upgradeable.name() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#67-69)
symbol() should be declared external:
        - ERC20Upgradeable.symbol() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#75-77)
decimals() should be declared external:
        - ERC20Upgradeable.decimals() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#92-94)
totalSupply() should be declared external:
        - ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#99-101)
balanceOf(address) should be declared external:
        - ERC20Upgradeable.balanceOf(address) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#106-108)
transfer(address,uint256) should be declared external:
        - ERC20Upgradeable.transfer(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#118-122)
approve(address,uint256) should be declared external:
        - ERC20Upgradeable.approve(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#141-145)
transferFrom(address,address,uint256) should be declared external:
        - ERC20Upgradeable.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#163-172)
increaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#186-190)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#206-215)
initialize() should be declared external:
        - PaymentVault.initialize() (contracts/PaymentVault.sol#38-40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Closing Summary

In this report, we have considered the security of the DappLooker codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, and Informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Important note to users

Approvals happen in a separate transaction from deposits. Users are advised to verify the amount of tokens approved for deposits.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the DappLooker Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the DappLooker Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
May, 2023

For

DappLooker

QuillAudits

Canada, India, Singapore, United Kingdom

www.quillaudits.com

audits@quillhash.com