



SOLIDProof

Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY

Lock Pay

AUDIT

SECURITY ASSESSMENT

17. July, 2023

FOR



LockPay



SolidProof_io



@solidproof_io

Introduction	3
Disclaimer	3
Project Overview	4
Summary	4
Social Medias	4
Audit Summary	5
File Overview	6
Imported packages	6
Audit Information	7
Vulnerability & Risk Level	7
Auditing Strategy and Techniques Applied	8
Methodology	8
Overall Security	9
Medium or higher issues	9
Upgradeability	10
Ownership	11
Ownership Privileges	12
Minting tokens	12
Burning tokens	13
Blacklist addresses	14
Fees and Tax	15
Lock Contract's functions and funds	16
Components	17
Exposed Functions	17
Capabilities	18
Inheritance Graph	19
Centralization Privileges	20
Audit Results	22

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	LockPay
Website	https://admin.lockpay.blockchaindevshop.com/
About the project	LockPay, a groundbreaking Web3 platform that seamlessly integrates the power of decentralized lockers and direct payment options to revolutionize the way businesses secure assets and accept cryptocurrency payments.
Chain	TBA
Language	Solidity
Codebase Link	https://github.com/Blockchain-Dev-Shop/Locker-SC/tree/develop
Commit	37ee41e
Unit Tests	Not Provided

Social Medias

Telegram	N/A
Twitter	N/A
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	17. July 2023	<ul style="list-style-type: none"> • Layout Project • Automated- /Manual-Security Testing • Summary

Note - This Audit report consists of a security analysis of the **LockPay** smart contracts. 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/ LockPayFactory.sol	0b0d3a86d8264040d7bacb3414ca0a69956473f0
contracts/Adminable.sol	e7226d788d9cee7c91de2b3e42c58d0abce49888
contracts/ LockPayVesting.sol	675940ca261004bcb279d943cf5095027e307b96
contracts/ LockPayGenerator.sol	85fd8def0e4a1427cc360d802874317621dbc657
contracts/ LockPaySettings.sol	9a9fc06312326592059f36edc3139608c5665c70

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/AccessControl.sol	1
@openzeppelin/contracts/access/Ownable.sol	5
@openzeppelin/contracts/security/ReentrancyGuard.sol	2
@openzeppelin/contracts/token/ERC20/IERC20.sol	3
@openzeppelin/contracts/utils/math/SafeMath.sol	3
@openzeppelin/contracts/utils/structs/EnumerableSet.sol	3

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 as possible.
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

No crucial Issues found

✓ Contract is safe to deploy

Description

The contract does not contain issues of high or medium criticality. This means that no known vulnerabilities were found in the source code.

Comment

It is to be noted that any user can create a lock for any arbitrary user.



Upgradeability

Contracts are not an upgradeable



Deployer cannot update the contracts 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

The contracts are not directly upgradeable but the owner is able to change the factory and generator addresses.

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, 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 set fees greater than 25%

✗ The owner able to levy unfair taxes

Description	The owner is not able to set the fees above 30%, but the early withdrawal penalty fees can be upto 50%
Comment	The lock fee, relock fee, and referral fees can be set up to 30% only



Lock Contract's functions and funds

In a smart contract, locking refers to restricting access to certain tokens or assets for a specified period. 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 can lock functionalities and funds	✗ The owner is able to lock the contract's functionality and funds
Description	Locking the contract means that the owner is able to lock any functions that may restrict the users from using them
Example	An example of locking is by pausing the contract's lock, relock, and transfer lock ownership functions. If the owner manually pauses these functions then no user will be able to lock their tokens or transfer the lock's ownership. Moreover, the owner is able to set the max withdraw percentage to any arbitrary value including very close to zero and if done so then no one would be able to withdraw any funds.
Comment	<ol style="list-style-type: none"> 1. The lockPayVesting contract can be paused. 2. The owner can change the generator address to an EOA in the factory contract and if done so then the users won't be able to register their lock anymore

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	0	0	1


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
56	1





External	Internal	Private	Pure	View
54	34	0	0	25

StateVariables

Total	 Public
26	20



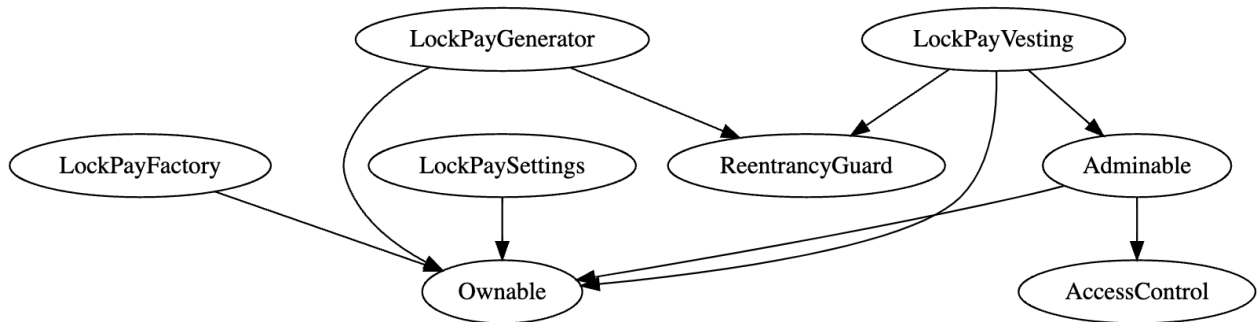
Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
<code>^0.8.0</code> <code>^0.8.4</code>	-----	----	----	-----



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. LockPayVesting.sol	<ul style="list-style-type: none"> ❖ onlyGenerator <ul style="list-style-type: none"> - Initialize the contract - Add duration only after the contract is initialized - Set early withdraw penalty to any value including 100% or more - Set refund settings - Add referral address ❖ onlyOwnerOrAdmin <ul style="list-style-type: none"> - Update settings of the contract - Pause/Unpause functions - Update refund settings ❖ onlyOwner <ul style="list-style-type: none"> - Add migrator address - Include/Exclude wallets from fees' whitelist
2. LockPaySettings.sol	<ul style="list-style-type: none"> ❖ onlyOwner <ul style="list-style-type: none"> - Update factory and Generator address - Set fee token, fee beneficiary and creation fees - Add/Remove allowed referrer addresses - Update default fees
3. LockPayFactory.sol	<ul style="list-style-type: none"> - The owner can change the generator address to an EOA and if done so then the users won't be able to register their lock anymore

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 | The penalty fees can be 50% or more

File	Severity	Location	Status
LockPayVesting	Medium	L156	Fixed

Description - The owner is able to change the early withdrawal penalty fees to any arbitrary value including 100% which will result in the loss of user funds if one tries to withdraw early.

Remediation - We recommend putting a reasonable hard cap on penalty fees.

Alleviation- The LocPay team has confirmed that this is part of their business logic.

#2 | Owner can lock funds

File	Severity	Location	Status
LockPayVesting	Medium	L163	Open

Description - The owner of the contract is able to lock the withdraw functionality by setting the maxWithdrawPercentage value to a very low number just above zero

#3 | Missing Zero Address Validation

File	Severity	Location	Status
LockPayVesting	Low	L231	Open

Description

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

#4 | Missng "isContract" check

File	Severity	Location	Status
LockPayGenerator	Low	L54	Open

Description

- The contract doesn't have any checks to verify whether the factory function is an EOA or a contract.

Remediation - We recommend putting in a check to verify that the factory address must be in a contract.

Note - This same bug also exists in the LockPaySettings contract on the line90

#5 | NatSpec documentation missing

File	Severity	Location	Status
All	Informational	—	Open

Description

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

#6 | Floating Pragma

File	Severity	Location	Status
All	Informational	L1	Open

Description

- The current pragma Solidity directives are "`^0.8.0` and "`^0.8.4`". 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.

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

File	Severity	Location	Status
All	Informational	N/A	Open

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.



**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY