SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Crypto Fixe

AUDIT

SECURITY ASSESSMENT

17. August, 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	CryptoFixe
Website	https://cryptofixe.com
About the project	CryptoFixe is a registered trademark in Portugal, which aims to revolutionize the DEFI and NFT market with simple and accurate solutions.
Chain	Arbitrum and Ethereum
Language	Solidity
Codebase Link	https://github.com/CryptoFixe/Smart-Contracts/blob/main/Token/CryptoFixe.sol
Commit	6ec3f78
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/CryptoFixe
Twitter	https://twitter.com/CryptoFixe_
Facebook	N/A
Instagram	https://www.instagram.com/CryptoFixe/
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	08. August 2023	• Layout Project • Automated- /Manual-Security Testing • Summary
v1.1	17. August 2023	• Reaudit

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

File Overview

The 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/CryptoFixe.sol	ca46bb5a07cb712d184e2e480fadd950e33702e4

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).

N/A

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

Ownership

The ownership is not renounced	❌ **The owner is not renounce**
Description	The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including: • Centralizations • The owner has significant control over contract's operations
Comment	If the ownership is renounced then no function can be called with the onlyBridge and onlyAmin modifiers.

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 can mint new tokens	❌ The owner able to mint new tokens
Description	Owners who have the ability to mint new tokens can reward themselves or other stakeholders, who can then sell the newly minted tokens on a cryptocurrency exchange to raise funds. However, there is a risk that the owner may abuse this power, leading to a decrease in trust and credibility in the project or platform. If stakeholders perceive that the owner is using their power to mint new tokens unfairly or without transparency, it can result in decreased demand for the token and a reduction in its value.
Example	If investors drive up the token price, the owner may choose to mint new tokens and sell them on a cryptocurrency exchange to raise funds. If the owner is not transparent and honest about their actions, they may be attempting a rugpull, where they suddenly abandon the project after raising funds, leaving investors with worthless tokens. This can lead to a decrease in the value of existing tokens, potentially rendering them worthless, and causing investors to suffer losses. It is essential for investors to carefully research the project and its developers and exercise caution before investing in any cryptocurrency or DeFi project.
Comment	The bridge address can mint tokens till the max supply is reached

Codebase -

```
321     function mint(address account↑, uint256 amount↑) external onlyBridge {
322         require((totalSupply() + amount↑) <= (MAX_SUPPLY * 10**DECIMALS) ,"TradeManagedToken:
323         _mint(account↑, amount↑);
324     }
```

Burning tokens

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

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

Blacklist addresses

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

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

Fees and Tax

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

Contract owner cannot set fees more than 0.05%	✅ The owner cannot levy unfair taxes
Description	The owner is not able to set the fees above 0.05%
Comment	N/A

Lock User Funds

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

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

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
2	2	6	4

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
53	1

External	Internal	Private	Pure	View
36	98	7	1	30

StateVariables

Total	🌐 Public
26	15

Capabilities

Solidity Versions observed	📤 Transfers ETH	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
^0.8.21	Yes	Yes	————	————

Inheritance Graph

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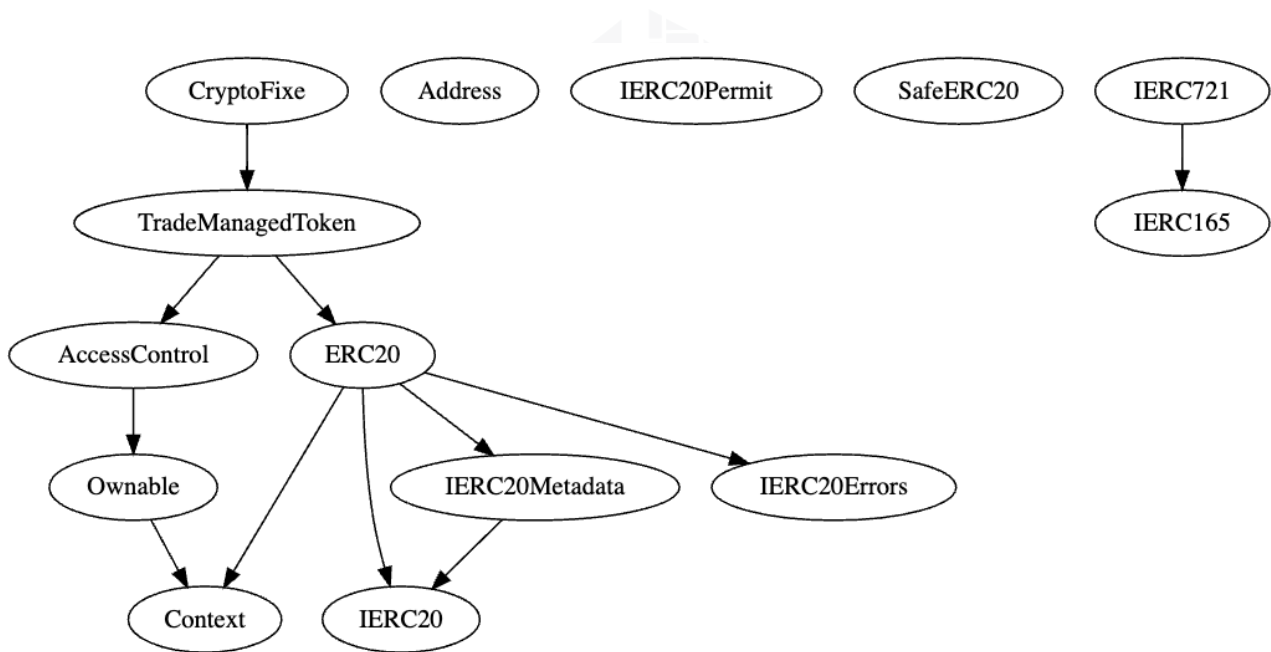
Centralization Privileges

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

CryptoFixe.sol	– onlyOwner
	   • Add/Remove bridge and admin addresses. This can be set as both EOA or contracts because there is no check.
	   • Admins can renounce their authority only before the owner is not renounced.
	   • Enable Trading
	   • Claim stuck tokens from the contract, but not the native ones
	   • Set LP pair
	   • Include/Exclude wallets from fees
	   • Set Rewards, Marketing, and Liquidity Address
	   • Set Max fees but not more than 0.05%
	   • Enable/Disable Especial NFT fees
	– onlyAdmin
	   • Process Fee Reserves
	– onlyBridge
	   • Mint tokens till the max supply is reached
	   • Burn tokens from any account without allowance

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

Critical issues

No critical issues

High issues

No high issues

Medium issues

#1 | Owner can Mint

File	Severity	Location	Status
Main	Medium	L305	ACK

Description - The owner can set the Bridge address as their own and then mint tokens till the max supply is reached

Alleviation - *It's part of the strategy mentioned earlier to control the supply between networks using a bridge and always maintain liquidity in circulation. Our primary strategy is to have a circulating supply with guaranteed liquidity and never exceed the maximum supply amount.*

#2 | Owner can burn tokens

File	Severity	Location	Status
Main	Medium	L326	Fixed

Description - The owner of the contract is able to burn tokens from any arbitrary wallet without any allowance by setting the bridge address as one of their own

Remediation - Make sure that it is not possible for any authority to burn tokens without allowance.

#3 | Tokens can be traded before the trade is active

File	Severity	Location	Status
Main	Medium	L296	ACK

Description - The trading needs to be enabled by the owner in order for regular users to transfer tokens. On the contrary, the owner can authorize addresses as admins manually and those addresses will be able to trade tokens. This functionality can be exploited in the following way, For example, there is a presale and the wallets used for the presale can be authorized by the owner. All the tokens obtained can be consolidated into a final wallet address and facilitate trading and selling of the acquired tokens, the last wallet address can be authorized.

Alleviation - "*Trading will not be allowed until liquidity is added after the public presale on GemPad. Only the wallet with presale tokens will have permission for transfers, making the token claim possible. After the presale and liquidity are added to the sushiswap, we will enable trading.*"

![SOLIDProof logo]

Low issues

#1 | Missing Events

File	Severity	Location	Status
Main	Low	L90—105, 288,	Fixed

Description - Make sure to emit events for all the critical parameter changes in the contract to ensure the transparency and trackability of all the state variable changes.

#2 | Missng "isContract" check

File	Severity	Location	Status
Main	Low	L100	ACK

Description - The contract has no checks to verify whether an EOA or a contract is set as an admin or bridge address.

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

Alleviation - *We have added the isContract verification in the addAdmin function, but we cannot do the same in the addBridge function. This is because we might have a contract managing the bridge, so we kept it without the verification.*

Informational issues

#1 | NatSpec documentation missing

File	Severity	Location	Status
Main	Informational	—	ACK

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

#2 | Floating Pragma

File	Severity	Location	Status
Main	Informational	L17	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.

Alleviation - *The deployment will take place right after the audit, and yes, we will ensure that it's done using the version mentioned in the contract, ^0.8.21*

#3 | 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.

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

MADE IN GERMANY

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY