SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY

BTCX

AUDIT

SECURITY ASSESSMENT

14. August, 2023

FOR

BTCX

SolidProof_io @solidproof_io

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	BTCX
Website	BTCX.io
About the project	The impact of Bitcoin on other cryptocurrencies is apparent. Bitcoin, like any other technology, has the potential to be improved. BTCX Token was created to enhance Bitcoin. We are using Bitcoin's advantages and offering next-generation functionality by merging hyper deflationary features with additional utilities such as staking and our own blockchain.
Chain	Ethereum
Language	Solidity
Codebase Link	https://github.com/btcx-token/BTCXTOKEN_CONTRACT/blob/main/BTCX.sol
Commit	d959beb01095c6edc61e886a5903466d9b3430c3
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/btcx_token
Twitter	https://twitter.com/btcx_token
Facebook	N/A
Instagram	N/A
Github	https://t.me/btcx_token
Reddit	N/A
Medium	N/A
Discord	N/A
Youtube	N/A
TikTok	N/A
LinkedIn	N/A

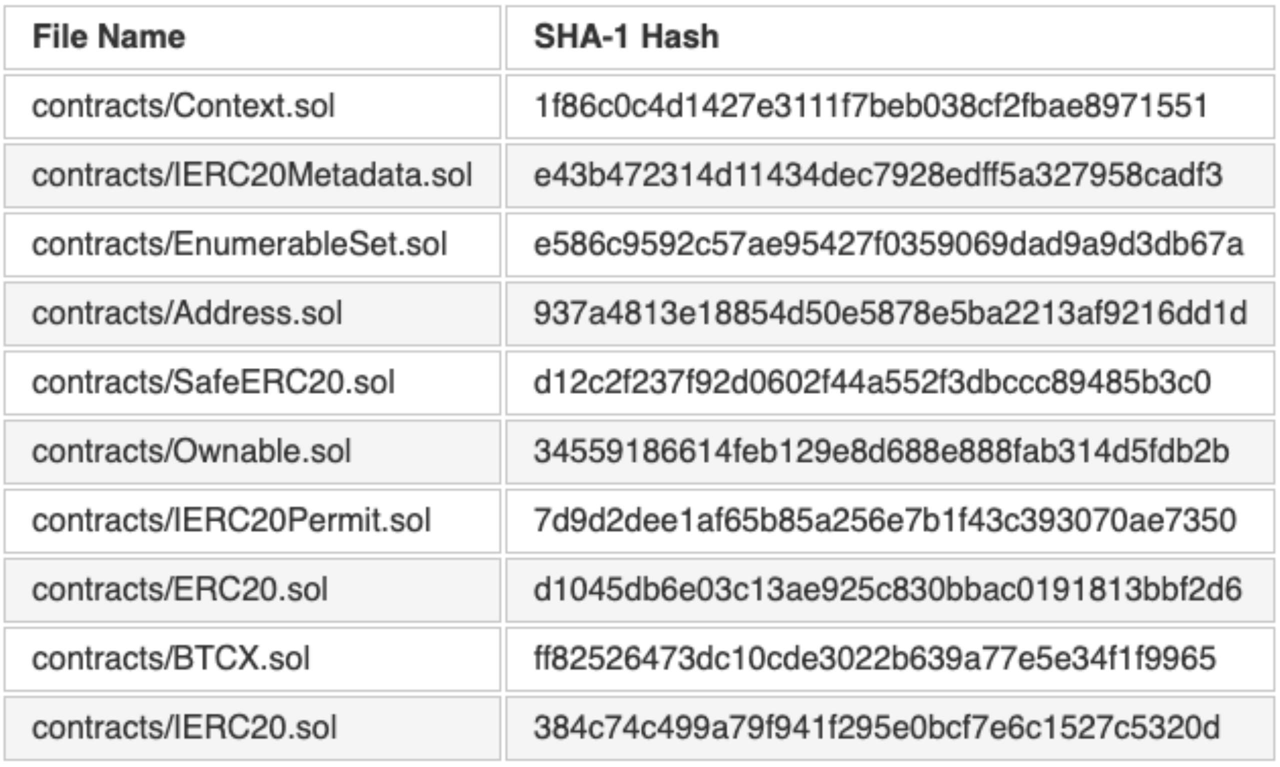
Audit Summary

Version	Delivery Date	Changelog
v1.0	14. August 2023	• Layout Project • Automated- /Manual-Security Testing • Summary

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.

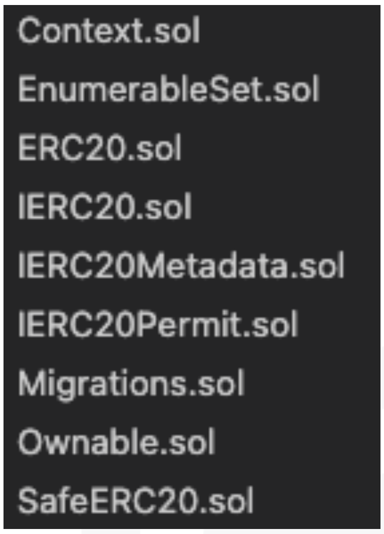


File Name	SHA-1 Hash
contracts/Context.sol	1f86c0c4d1427e3111f7beb038cf2fbae8971551
contracts/IERC20Metadata.sol	e43b472314d11434dec7928edff5a327958cadf3
contracts/EnumerableSet.sol	e586c9592c57ae95427f0359069dad9a9d3db67a
contracts/Address.sol	937a4813e18854d50e5878e5ba2213af9216dd1d
contracts/SafeERC20.sol	d12c2f237f92d0602f44a552f3dbccc89485b3c0
contracts/Ownable.sol	34559186614feb129e8d688e888fab314d5fdb2b
contracts/IERC20Permit.sol	7d9d2dee1af65b85a256e7b1f43c393070ae7350
contracts/ERC20.sol	d1045db6e03c13ae925c830bbac0191813bbf2d6
contracts/BTCX.sol	ff82526473dc10cde3022b639a77e5e34f1f9965
contracts/IERC20.sol	384c74c499a79f941f295e0bcf7e6c1527c5320d

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).



```
Context.sol
EnumerableSet.sol
ERC20.sol
IERC20.sol
IERC20Metadata.sol
IERC20Permit.sol
Migrations.sol
Ownable.sol
SafeERC20.sol
```

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
Example	We assume that the ownership is not renounced or there are any other authorities. Now imagine that the owner/authorities can toggle off the e.g. claiming function from a contract to lock the function. However, the owner/authorities are able to modify the contract while it is deployed to change the behaviour the contract.
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.

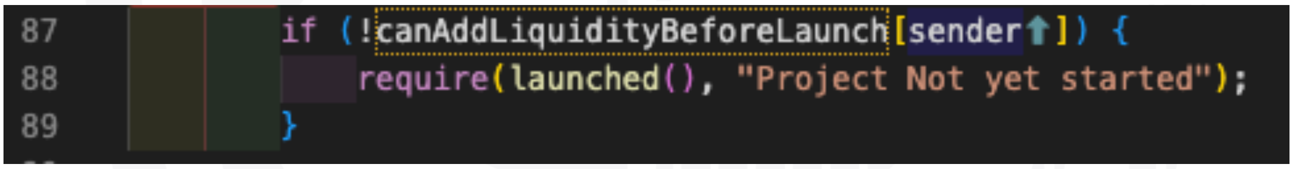
Contract owner cannot set fees more than 25%	✅ **The owner cannot levy unfair taxes**
Description	The owner is not able to set the fees above 25%
Comment	Buy and sell burn fee is set to 1%

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	After the launch the owner cannot lock user funds. Remember that the owner can add presale wallets which can transfer tokens without launching the contract.

File, Line/s: BTCX.sol, L87
Codebase:



```
if (!canAddLiquidityBeforeLaunch[sender]) {
    require(launched(), "Project Not yet started");
}
```

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	3	3	2

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
40	0

External	Internal	Private	Pure	View
16	109	8	2	44

StateVariables

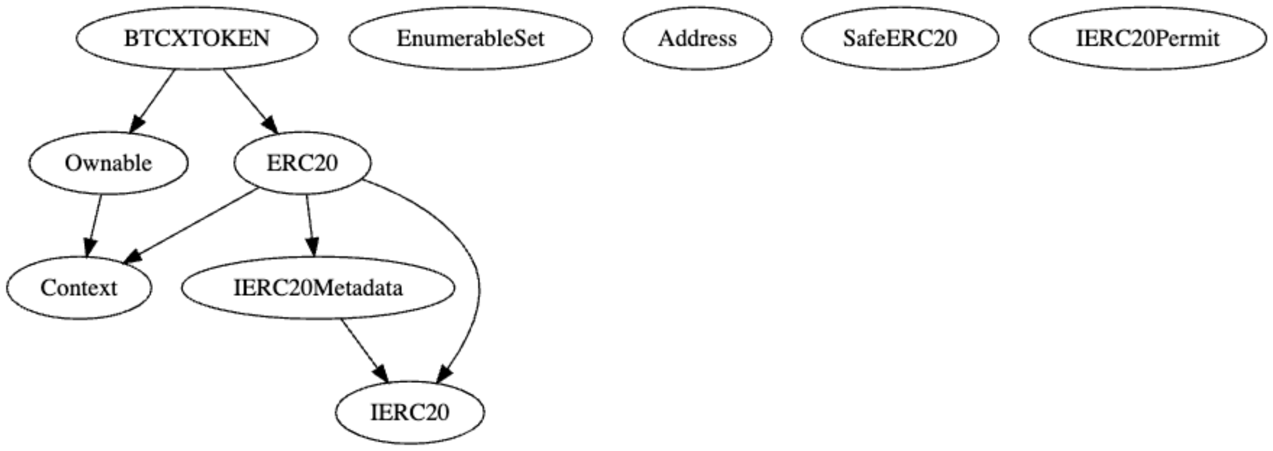
Total	🌐Public
21	12

Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.0` `^0.8.1` `=0.8.19`			`yes` (4 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. BTCX.sol	• OnlyOwner • launch • rescueToken • setIsFeeExempt • setPresaleWallet • setBurnSettings • addPair • delPair

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

No medium issues

Low issues

#1 | Owner is able to get out their own tokens

File	Severity	Location	Status
Main	Low	L162	Open

Description - The owner can get out their contract tokens.

Remediation - It is recommended to prevent passing the own contract address.

#2 | Shadowing local variable

File	Severity	Location	Status
Main	Low	L62	Open

Description - The "_totalSupply" is shadowing the ERC20._totalSupply variable.

Remediation - Changing the "_totalSupply" to "totalSupply_" is recommended.

#3 | Missing event

File	Severity	Location	Status
Main	Low	L183, L178, L174, L170, L167, L162, L156	Open

Description - Emit an event when a state variable is changed or modified.

Informational issues

#1 | NatSpec documentation missing

File	Severity	Location	Status
Main	Informational	—	Open

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

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

File	Severity	Location	Status
Main	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.

#3 | Variable can be declared as constant

File	Severity	Location	Status
Main	Informational	L46, L50, L43, L47, L51	Open

Description - Add the “constant” attribute to state variables that never change.

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