SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Token Quest

AUDIT

SECURITY ASSESSMENT

14. April, 2024

FOR

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	Token Quest
Website	https://tokenquest.org/
About the project	TokenQuest is a DeFi project on BSC with the goal of creating an ecosystem where users are incentivized through engaging quests and features to contribute to the platform and earn rewards.
Chain	BSC
Language	Solidity
Codebase Link	**tQUEST:** 0x7BC6174EC24903D5B8D79618cd89AE82ba1A956C **tGOLD:** 0x84503BB41b13a2CC74D006091863667D4e59760D
Fork Status	The smart cotnracts are inspired from the smart contracts of https://burnedfi.app/#/home
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/tokenquestorg
Twitter	https://twitter.com/tokenquestorg
Facebook	N/A
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	https://medium.com/@tokenquestorg
Discord	https://discord.gg/cuq6bCgdpd
Youtube	https://www.youtube.com/@tokenquestorg
TikTok	N/A
LinkedIn	N/A

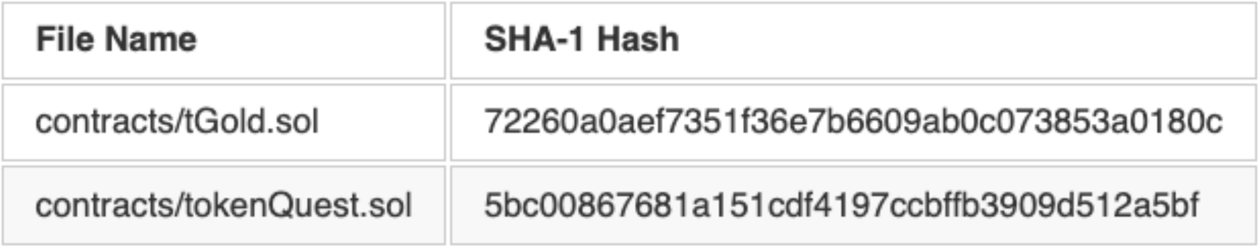
Audit Summary

Version	Delivery Date	Changelog
v1.0	14. April 2024	• 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, including malicious outside manipulation of the contract's functions. 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 we did not functionally test it. This includes internal calculations in the formulae used in the contract.

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/tGold.sol	72260a0aef7351f36e7b6609ab0c073853a0180c
contracts/tokenQuest.sol	5bc00867681a151cdf4197ccbffb3909d512a5bf

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) indicate 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 the 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. Review the specifications, sources, and instructions provided to SolidProof to ensure we understand the smart contract's size, scope, and functionality.
    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 is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., smart contracts, to improve efficiency, effectiveness, clarity, maintainability, security, and control based on industry and academia's best practices, recommendations, and research.

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	Both the cotnract addresses can be changed by the owner and those new contrats may or may not contain new functions that may or may not harm users.

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, we would consider ownership to be unrenounced. 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 refers 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 can 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 without allowance	✅ **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.

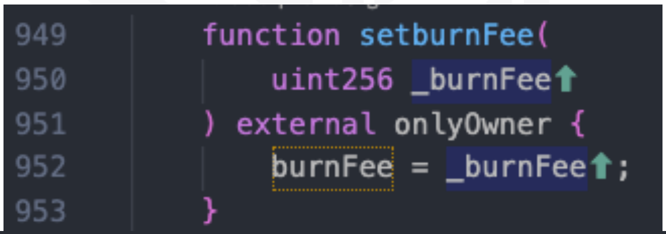
Contract owner cannot blacklist addresses	✅ **The owner cannot blacklist addresses**
Description	The owner is not able blacklist addresses to lock funds.
Comment	N/A

Fees and Tax

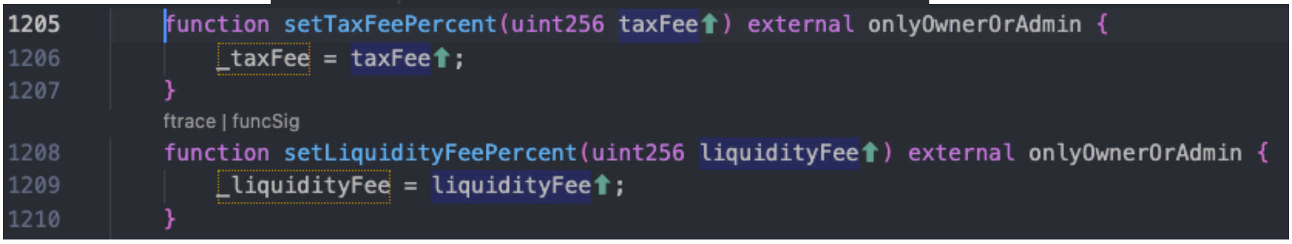
In some smart contracts, the owner or creator can set fees for certain actions or operations within the contract. These fees can be used to cover the contract's cost, such as paying gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner can set fees greater than 25%	❌ **The owner able to burn tokens**
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. 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 observation is that the owner can adjust the burn fees and other fees up to 100%. If the fee is set to 100%, it implies that the full amount of tokens you intend to send will either be burned or the owner can also withdraw them from 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 fee. However, the same is applicable for the tax fee and liquidity fee
Comment	N/A

File: tokenQuest.sol, tGold.sol
Codebase:



```
    function setburnFee(
        uint256 _burnFee
    ) external onlyOwner {
        burnFee = _burnFee;
    }
```



```
function setTaxFeePercent(uint256 taxFee) external onlyOwnerOrAdmin {
    _taxFee = taxFee;
}
ftrace | funcSig
function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwnerOrAdmin {
    _liquidityFee = liquidityFee;
}
```

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.

Contract owner can lock the user funds	❌ **The owner is able to lock the contract**
Description	Locking the contract means that the owner is able to lock any funds of addresses that they are not able to transfer bought tokens anymore.
Example	An example of locking is by setting the fees to 100%
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
5	4	13	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
203	13

External	Internal	Private	Pure	View
158	173	32	38	94

StateVariables

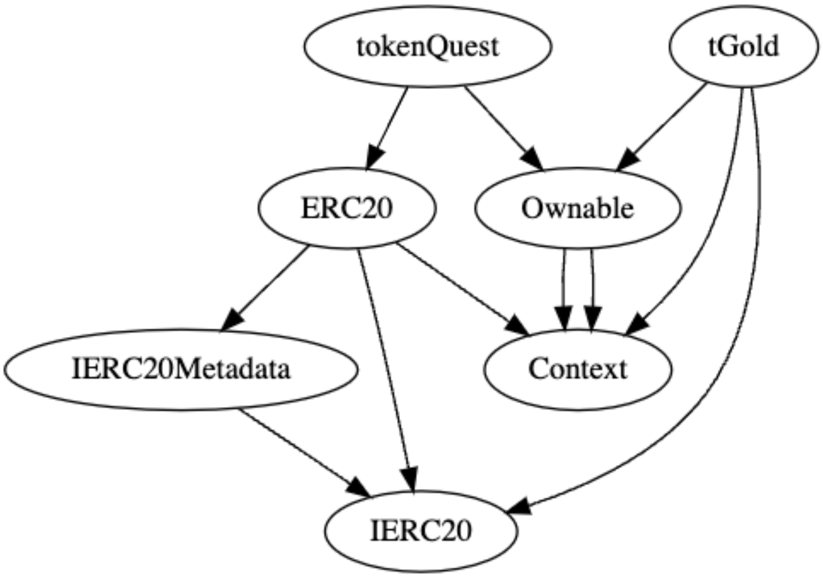
Total	🌐Public
59	27

Capabilities

Solidity Versions observed	Transfers ETH	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.0`	Yes	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 a single entity controls the contract or if certain participants have special permissions or abilities that others do not.

In the project, some authorities have access to the following functions:

File	Privileges
tokenQuest.sol	• Set token gold address • Set Pair, AMM, and launcher address • Set burn fee to an arbitrary value • Include/Exclude wallets from fees
tGold.sol	• Add/Remove admins • Only the owner or admin can transfer tokens • Include/Exclude wallets from fees • Set fees to an arbitrary value • Set/Update tokenQuest contract address

Recommendations

To avoid potential hacking risks, the client should 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 can no longer modify any state variables of the contract. Make sure to set up everything before renouncing.

Audit Results

Critical issues

No critical issues

High issues

No high issues

Medium issues

#1 | Reentrancy Risk

File	Severity	Location	Status
tGold	Medium	L1376	ACK

Description — The function violates the "Check, Effects, and Interaction pattern," which leads to reentrancy risk. The transaction could run out of gas because "call" was used, but the risk of reentrancy persists.

When native tokens are sent, the receive() function in the receiving contract is activated. This opens up the possibility for an attacker to execute a recursive call back to the initial function. In this scenario, state variables remain unaltered during the re-entrant call (as the caller still possesses the tGold token), allowing the attacker to re-enter the original contract and potentially claim rewards multiple times.

Remediation — We recommend using the non-reentrant modifier by Openzeppelin and properly following the Check-effects-interaction pattern in the code.

#2 | Owner can Lock User Funds (Centralization Risk)

File	Severity	Location	Status
tGold.sol	Medium	L1205, 1208	ACK
tokenQuest.sol	Medium	L949	ACK

Description—The contract owner can set the fees up to 100%. If so, the recipient will not receive any funds when transferring the tokens.

The owner can also set the burn fees to 100% or even more in the tokenQuest contract and then set the percentage for LP burn to zero. In that case, the tokens won't be burnt, and the funds collected as the marketing fees can easily be transferred to the owner's wallet by setting the "toknGold" address as their own and calling the "tokenToGold" function.

Remediation - Ensure the maximum fee limit is less than 25%. The burn and marketing fees should be treated differently, and the burn fees should only be used when burning the LP.

Low issues

#1 | Missing Events

File	Severity	Location	Status
All	Low	N/A	ACK

Description— Emit events for all the critical parameter changes in the contract to ensure the transparency and trackability of all the state variable changes. In the codebase, we have observed that only a few state-changing functions emit events. We suggest that all the state-changing functions that are user-controlled and owner-controlled emit their respective events.

#2 | Possible Flashloan Attack (Missng "isContract" check)

File	Severity	Location	Status
Main	Low	L1411	ACK

Description— The contract depends on a single oracle to get the token's on-chain price, making it susceptible to a flash-loan attack, in which the attacker can manipulate the token price on a DEX for personal gain.

Remediation— We recommend restricting the function to be called by an EOA only. This significantly reduces the risk of a FlashLoan attack. If that's not possible, then Time-Weighted Average Price (TWAP) should be used for the price reference. However, using TWAP, the price manipulated on one block does not significantly impact the average price.

Informational issues

#1 | NatSpec documentation missing

File	Severity	Location	Status
All	Informational	N/A	ACK

Description - If you have 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
All	Informational	N/A	ACK

Description— We recommend importing all packages from npm directly without flattening the contract. Functions could be modified or 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

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY