QuillAudits

Audit Report
July, 2023

For

TechBank

Table of Content

Executive Summary 02

Checked Vulnerabilities 04

Techniques and Methods 05

Manual Testing 06

A. Contract - TechBank.sol 06

B. Contract - IERC20.sol 10

Functional Tests 11

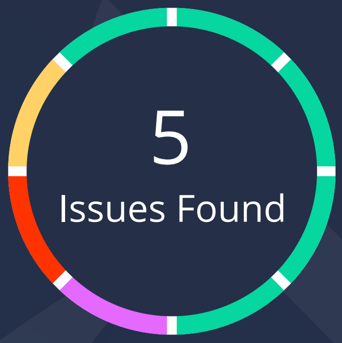
Automated Tests 11

Closing Summary 12

About QuillAudits 13

Executive Summary

Project Name	TechBank
Overview	The TechBank project aims to provide the functionality of a bank by allowing deposits, collateral-backed loans, and interest to users.
Timeline	5th July, 2023 to 7th July, 2023
Method	Manual Review, Automated Testing, Functional Testing, etc.
Audit Scope	The scope of this audit was to analyze TechBank codebase for quality, security, and correctness. *https://drive.google.com/drive/folders/1z1z50_30cSFj4qucA_o2DwddlFTrfCyr*
Contracts in Scope	- TechBank.sol - IERC20.sol
Fixed In	*https://drive.google.com/drive/folders/1ZUAjXQwsiwz9hgmGL6svzFlEOy1_UMA7*



🟥 High 🟨 Medium

🟩 Low 🟪 Informational

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

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 - TechBank.sol

High Severity Issues

A.1 Interest rates

Description

Although there is no defined process for interest generation (there is a lack of proper documentation), the protocol allows generated interest to be withdrawn only before the 7-day window elapses. This is what the current implementation allows due to the check present in the if statement

```
function getInterest(address account) public view returns (uint256) {
  ...                                    // 7 days after deposit time
  if (depositTime == 0 || block.timestamp >= depositTime + 7 days) {
    return 0;
  }
}
```

If depositTime is zero or the current block.timestamp is higher than 7 days after the initial depositTime then 0 is returned, when 0 is returned to withdrawInterest, the call reverts. This puts the maximum interest rate to be ~7% just before the 7-day timer elapses due to the 1% daily interest accrued. The risk is interest gets locked forever after passing the 7-day mark.

Remediation

Properly document code, apply in-line comments, and ensure users thoroughly understand the workings of the protocol before use.

Status

Resolved

Medium Severity Issues

A.2 Hardcoded addresses

Description

For protocols that are linked/dependent on other contracts for their functionality (i.e. payments, staking, governance), there is a high risk of incurring a denial of service if those contracts and exploited or updated by their maintainers. TechBank has the BSC-USDT address hardcoded in the constructor with no possibility to update it via a protected setter function. This is risky as the entire protocol does not provide a fallback if the USDT contract is unavailable.

Also, if the TechBank contract is to be deployed across multiple chains, it is highly unlikely that the same address on all chains points to the USDT contract, it could be a random EOA or contract without the expected ERC20 functionality which can brick the protocol.

Remediation

1. Provide a fallback functionality to the TechBank contract by means of a setter function.
2. Pass the usdtAddress as a parameter in the constructor instead of hardcoding the address.

Status

Acknowledged

Low Severity Issues

A.3 Invalid checks

Description

In lend(), the require statement checks for available collateral and not ‘amount + collateral’. The transferFrom() call can revert even after passing the collateral require check. This could lead to increased gas costs for users on failed transactions due to the external call to the USDT contract.

Remediation

Include the total value check (amount + collateral) to ensure that users have sufficient balance, and for gas efficiency.

Status

Resolved

A.4 Missing test cases

Description

Unit tests are used to ensure that the code functions as expected by passing in varying user input and setting up various parameters to test the boundaries of the protocol. There are no unit test cases associated with the codebase provided, hereby increasing the probability of bugs being present and reducing quality assurance.

Remediation

Include unit tests that have > 95% code coverage including all possible paths for code execution.

Status

Acknowledged

Informational Issues

A.5 Unlocked pragma (pragma solidity >=0.8.0)

Description

Contracts should be deployed with the same compiler version that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated/newly released compiler version that might introduce bugs that could affect the contract system negatively.

Remediation

It is recommended to lock the solidity pragma contract to a specific version.

Status

Acknowledged

B. Contract - IERC20.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

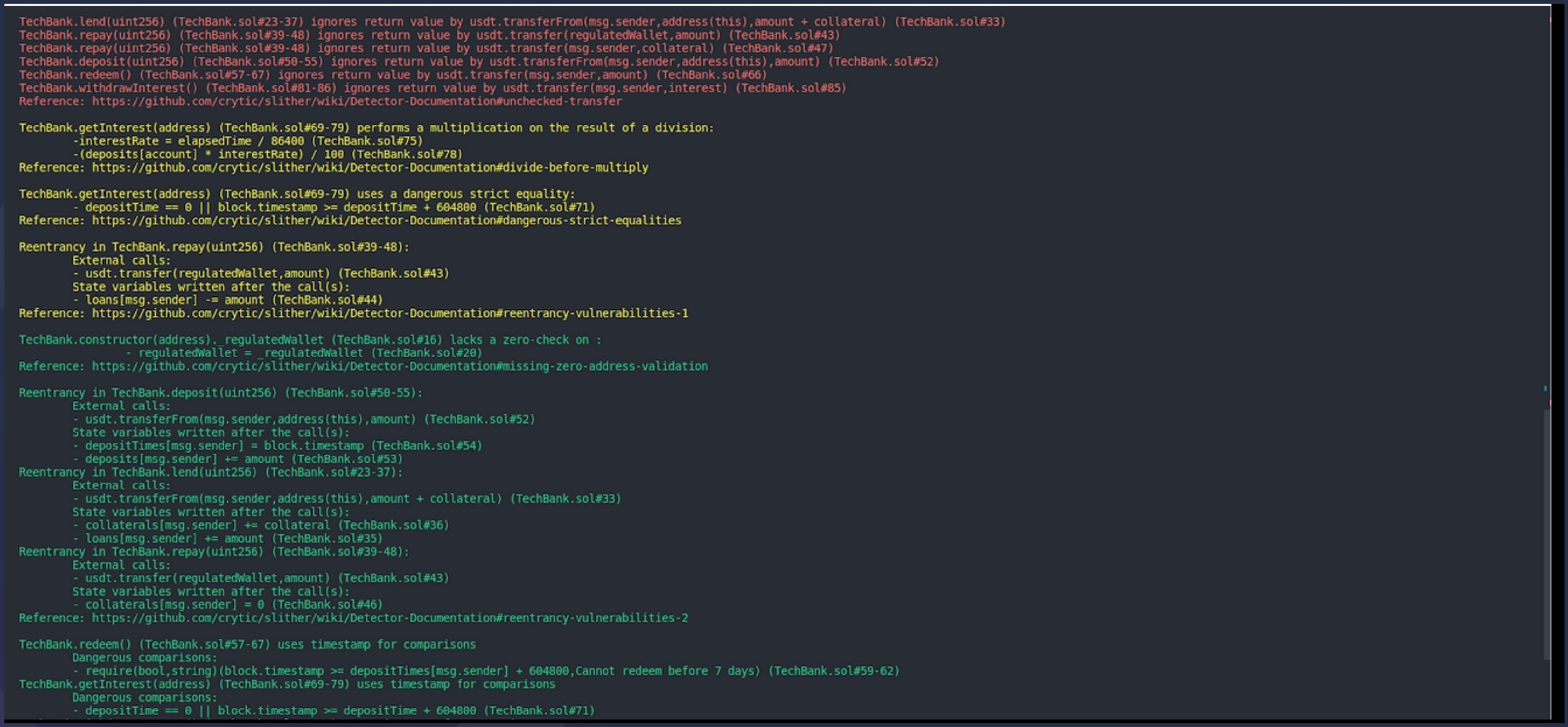
Functional Testing

Some of the tests performed are mentioned below:

- ✓ lend(): Should check for collateral and amount in user's balance
- ✓ Should revert on failed transfer and transferFrom calls.
- ✓ Should not increase loans, collateral mapping on failed loan() calls.
- ✓ Should not lend tokens to accounts with no collateral. [FAILED]
- ✓ Should allow interest to be withdrawn after the 7-day lockup period.

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.



```
TechBank.lend(uint256) (TechBank.sol#23-37) ignores return value by usdt.transferFrom(msg.sender,address(this),amount + collateral) (TechBank.sol#33)
TechBank.repay(uint256) (TechBank.sol#39-48) ignores return value by usdt.transfer(regulatedWallet,amount) (TechBank.sol#43)
TechBank.repay(uint256) (TechBank.sol#39-48) ignores return value by usdt.transfer(msg.sender,collateral) (TechBank.sol#47)
TechBank.deposit(uint256) (TechBank.sol#50-55) ignores return value by usdt.transferFrom(msg.sender,address(this),amount) (TechBank.sol#52)
TechBank.redeem() (TechBank.sol#57-67) ignores return value by usdt.transfer(msg.sender,amount) (TechBank.sol#66)
TechBank.withdrawInterest() (TechBank.sol#81-86) ignores return value by usdt.transfer(msg.sender,interest) (TechBank.sol#85)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

TechBank.getInterest(address) (TechBank.sol#69-79) performs a multiplication on the result of a division:
        -interestRate = elapsedTime / 86400 (TechBank.sol#75)
        -(deposits[account] * interestRate) / 100 (TechBank.sol#78)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

TechBank.getInterest(address) (TechBank.sol#69-79) uses a dangerous strict equality:
        - depositTime == 0 || block.timestamp >= depositTime + 604800 (TechBank.sol#71)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in TechBank.repay(uint256) (TechBank.sol#39-48):
        External calls:
        - usdt.transfer(regulatedWallet,amount) (TechBank.sol#43)
        State variables written after the call(s):
        - loans[msg.sender] -= amount (TechBank.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

TechBank.constructor(address)._regulatedWallet (TechBank.sol#16) lacks a zero-check on :
                - regulatedWallet = _regulatedWallet (TechBank.sol#20)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in TechBank.deposit(uint256) (TechBank.sol#50-55):
        External calls:
        - usdt.transferFrom(msg.sender,address(this),amount) (TechBank.sol#52)
        State variables written after the call(s):
        - depositTimes[msg.sender] = block.timestamp (TechBank.sol#54)
        - deposits[msg.sender] += amount (TechBank.sol#53)
Reentrancy in TechBank.lend(uint256) (TechBank.sol#23-37):
        External calls:
        - usdt.transferFrom(msg.sender,address(this),amount + collateral) (TechBank.sol#33)
        State variables written after the call(s):
        - collaterals[msg.sender] += collateral (TechBank.sol#36)
        - loans[msg.sender] += amount (TechBank.sol#35)
Reentrancy in TechBank.repay(uint256) (TechBank.sol#39-48):
        External calls:
        - usdt.transfer(regulatedWallet,amount) (TechBank.sol#43)
        State variables written after the call(s):
        - collaterals[msg.sender] = 0 (TechBank.sol#46)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

TechBank.redeem() (TechBank.sol#57-67) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= depositTimes[msg.sender] + 604800,Cannot redeem before 7 days) (TechBank.sol#59-62)
TechBank.getInterest(address) (TechBank.sol#69-79) uses timestamp for comparisons
        Dangerous comparisons:
        - depositTime == 0 || block.timestamp >= depositTime + 604800 (TechBank.sol#71)
```

Summary

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

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

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Techbank smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Techbank smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur as a result of using our audit services. It is the responsibility of the Techbank to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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.

850+	**$30B**	**800K**
Audits Completed	Secured	Lines of Code Audited

Follow Our Journey

Audit Report
July, 2023

For

TechBank

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com