QuillAudits

# Audit Report
# September, 2023

For

Sharp Token

# Table of Content


Executive Summary ..... 03

Number of security issues per severity ..... 04

Checked Vulnerabilities ..... 05

Techniques and Methods ..... 07

Types of Severity ..... 08

Types of Issues ..... 08

**A. Contract - TT10** 09

**High Severity Issues** 09

**Medium Severity Issues** 09

A.1: Centralization Risk Issue 09

**Low Severity Issues** 10

A.2: Use Two-way Process for Transfer of Ownership 10

**Informational Issues** 11

A.3: Inappropriate Comparison of Boolean 11

A.4: Missing beforeTransfer function in some critical functions 12

A.5: Unlocked pragma (pragma solidity ^0.8.18) 13

A.6: Use Natspec Comment Format 13

**B. General Recommendation** 14

# Table of Content


Functional Tests ..... 14

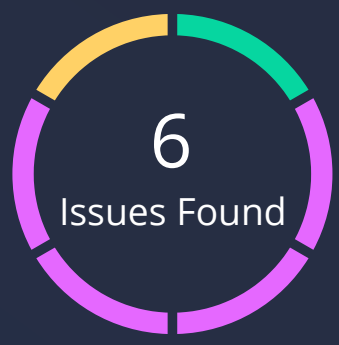
Automated Tests ..... 14

Closing Summary ..... 15

# Executive Summary

| | |
|---|---|
| **Project Name** | The Sharp Token |
| **Project URL** | *http://thesharptoken.com/* |
| **Overview** | TT10 contract by the Shark Token Team is an ERC20 token contract with the contract owner owning the permission to mint and burn the tokens. The contract inherits the Ownable, Pausable, and interafce of the ERC20 libraries from the standard Openzeppelin modules. With Ownable library, it modifies the access control to some functionalities to be only possible to be called by the contract owner. The Pausable library gives the contract owner the privilege to pause and unpause the major functionalities of the token. For instance, when the contract owner pauses the contract, users can no longer make transfer or call the transferFrom functions to spend approved tokens. Not until the contract owner unpause, will those activities work. |
| **Audit Scope** | https://goerli.etherscan.io/token/0xd5B189930C16Cb71cD7255d0Fc5dD4d6d0426106#code |
| **Contracts in Scope** | TT10 |
| **Commit Hash** | NA |
| **Language** | Solidity |
| **Blockchain** | Ethereum |
| **Method** | Manual Analysis, Functional Testing, Automated Testing |
| **Review 1** | 8th August 2023 - 10th August 2023 |
| **Updated Code Received** | 11th September 2023 |
| **Review 2** | 12th September 2023 |
| **Fixed In** | https://goerli.etherscan.io/address/0x05370C241A09c732E34fEfCac0E24DA4Ed8AbA82#code |
| **Mainnet Address** | https://polygonscan.com/token/0x03dB9D06bC596FAC5F963951032738Efcf51e328 |

# The Number of Security Issues per Severity




High
Medium
Low
Informational

| | High | Medium | Low | Informational |
|---|---|---|---|---|
| Open Issues | 0 | 0 | 0 | 0 |
| Acknowledged Issues | 0 | 0 | 1 | 0 |
| Partially Resolved Issues | 0 | 0 | 0 | 0 |
| Resolved Issues | 0 | 1 | 0 | 4 |

# Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

# Checked Vulnerabilities

- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

# Techniques and Methods

Throughout the audit of smart contracts, 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 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

A 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 analyzed, 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 behavior 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, Solhint, Mythril, Slither, Solidity statistic analysis.

## Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

### High Severity Issues

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 Severity Issues

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 Severity Issues

Low-level severity issues can cause minor impact and 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 four 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.

# A. Contract - TT10

## High Severity Issues

No issues were found

## Medium Severity Issues

### A.1: Centralization Risk Issue

**Description**

The ability of the contract owner to regulate the use of the token with the use of the Pausable library shows the centralization power centered on the contract owner. The owner of the contract can pause the contract and users will no longer be able to use these tokens again unless the owner unpauses the contract. When the contract is paused, the “beforeTransfer()” created in the Pausable returns the state of _paused variable. This is called in the following functions:
- transfer
- transferFrom
- decreaseAllowance

Also, with the presence of the TransferOwnership function in the token contract, it shows that the contract owner can transfer not just the ownership of the contract, but transfer the tokens present in the former contract address to the new contract address.

**Remediation**

Users should trust the owner of the contract and be assured that the pause and unpause of the contract is to their benefit.

**Status**

**Resolved**

Please Note: According to discussions with the TT10 team, the issue has been mentioned as Resolved; the TT10 team's statement is below.

**TT10 (The Sharp Token) team’s comment**

A multisig wallet was intended to be used as the contract owner.

## Low Severity Issues

### A.2: Use Two-way Process for Transfer of Ownership

```
function TransferOwnership(
    address addr⬆
) public virtual onlyOwner returns (bool) {
    require(addr⬆ != address(0), "Invalid address");
    _transfer(_msgSender(), addr⬆, _balances[_msgSender()]);
    transferOwnership(addr⬆);
    return true;
}
```

**Description**

In the token contract, there is the TransferOwnership function for the purpose of transferring ownership of the contract and all the tokens in the balance of the address from the old owner to the new address. One possibility here is, the owner can mistakenly pass in a wrong address, and this will cause losing ownership and all of the tokens to the wrong address and this is not irredeemable afterwards.

**Remediation**

It is recommended to use a two-way process for the ownership transfer and token transfer. This implies making an address the pending owner address until the new address accepts the ownership. This prevents sudden transfer. The Openzeppelin standard library also has the Ownable2Step module for this purpose.

**Status**

**Acknowledged**

**Reference**

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol

## Informational Issues

### A.3: Inappropriate Comparison of Boolean

```
require(beforeTransfer() == false, "Transfer Paused by owner");
```

**Description**

The beforeTransfer() function is a view function that returns a boolean and this is called in the require check of some critical function. The issue arises with comparing the returned value to the “false” constant.

**Remediation**

Remove the equality comparison and use the beforeTransfer function directly in the check.

**Status**

**Resolved**

**Reference**

https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

## A.4: Missing beforeTransfer function in some critical functions

```
function increaseAllowance(
    address spender,
    uint256 addedAmount
) external virtual returns (bool) {
    address owner = _msgSender();
    _approve(owner, spender, allowance(owner, spender) +
    addedAmount);
    return true;
}
```

```
function decreaseAllowance(
    address spender,
    uint256 requestedDecrease
) external virtual returns (bool) {
    require(beforeTransfer() == false, "Transfer Paused by owner");
    address owner = _msgSender();
    uint256 currentAllowance = allowance(owner, spender);
    if (currentAllowance < requestedDecrease) {
        revert("current allowance is less than requested decrease");
    }
    unchecked {
        _approve(owner, spender, currentAllowance -
        requestedDecrease);
    }

    return true;
}
```

### Description

According to the contract, the beforeTransfer function is used to pause critical functionalities such as transfer, transferFrom, and decreaseAllowance. However, this was not included in some other critical functions, like for instance, the direct opposite to the decreaseAllowance function, increaseAllowance, do not have the beforeTransfer check. This implies that when the contract is paused, users can still increase allowances of the approved addresses but they cannot decrease the allowances. Also another function is the approved function, this poses that token holders can still approve tokens to addresses. There is no essence of approving tokens that cannot be transferred.

### Remediation

Add the beforeTransfer function to the approve and increaseAllowance functions.

### Status

**Resolved**

## A.5: Unlocked pragma (pragma solidity ^0.8.18)

```
pragma solidity ^0.8.18;
```

### Description

Contracts should be deployed with the same compiler version and flags 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 compiler version that might introduce bugs that affect the contract system negatively.

### Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or third-party unaudited libraries. If necessary, refactor the current code base to only use stable features.

### Status

**Resolved**

## A.6: Use Natspec Comment Format

### Description

The codebase has comments on each of the functions. However, it fails to prototype the Natspec code comment format that details the functions more appropriately. With the use of the Natspec format, it is easier to cite the expected function inputs and explain them. This makes for easy comprehension of the codebase and is often recommended as the conventional way to comment solidity smart contracts.

### Remediation

Use the Natspec model of comment to ensure easy understanding of the codebase.

### Status

**Resolved**

### Reference

https://docs.soliditylang.org/en/v0.8.17/natspec-format.html

# B. General Recommendation

The TT10 token contract gives the contract owner a degree of privilege to mint, burn, and regulate the usage of the tokens. Therefore, it is pertinent to mention that the users of these tokens should have a high level of trust on who the contract owner would be because this contract owner can pause the activities of the token anytime and can unpause anytime.

# Functional Tests

**Some of the tests performed are mentioned below:**

- Should get the name of the token
- Should get the symbol of the token
- Should get the decimal of the token
- Should get the total supply of the token when owners mint to their addresses
- Should transfer tokens to other address
- Should approve another account to spend token
- Should mint into contract owner address and increase total supply
- Should allow the contract owner to pause the contract and halt major activities
- Should allow the contract owner to unpause and revive every critical activities
- Should revert when decreaseAllowance function is called when contract is paused

# 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.

# Closing Summary

In this report, we have considered the security of TT10 contract by the sharp token. We performed our audit according to the procedure described above.

No Issues found in the contract. Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the end, the Sharp Token team resolved almost all issues found and acknowledged one low severity issue.

# Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in TT10 smart contract. 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 TT10 smart contract. 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 subsequent to and despite using our audit services.. It is the responsibility of the TT10 (the sharp Token Team) 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+**
Audits Completed

**$30B**
Secured

**800K**
Lines of Code Audited

## Follow Our Journey

# Audit Report
# September, 2023

For

## Sharp Token

QuillAudits


Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com