QuillAudits

Audit Report
April, 2022

For

Bridge

Table of Content

Executive Summary

Project Name	Bridge Network
Overview	Bridge Token is a simple Token contract with extra blacklisting function. The blacklisting is controlled by admin address privilege
Timeline	April 11, 2022 - April 13, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Audit Scope	The scope of this audit was to analyze and document the Bridge Network's smart contract codebase for quality, security, and correctness. *https://github.com/bridgeNetwork1/bridgeToken/blob/master/bridgeToken.sol*
Branch	Master
Commit	9636c155e8fcfff5cda8b609a77c7d4527b5b402
Fixed in [version 1]	7f21f1e77497d83a97a466faa16c99b1a1a49357
Fixed in [version 2]	23ae222c422237b34759803793e5ada890ef2979

5
Issues Found

🟥 High 🟨 Medium

🟩 Low 🟪 Informational

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

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
- ✓ DoS with Block Gas Limit
- ✓ Transaction-Ordering Dependence
- ✓ Use of tx.origin
- ✓ Exception disorder
- ✓ Gasless send
- ✓ Balance equality
- ✓ Byte array
- ✓ Transfer forwards all gas
- ✓ ERC20 API violation
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Redundant fallback function
- ✓ Send instead of transfer
- ✓ Style guide violation
- ✓ Unchecked external call
- ✓ Unchecked math
- ✓ Unsafe type inference
- ✓ Implicit visibility leve

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 - BridgeToken.sol

High Severity Issues

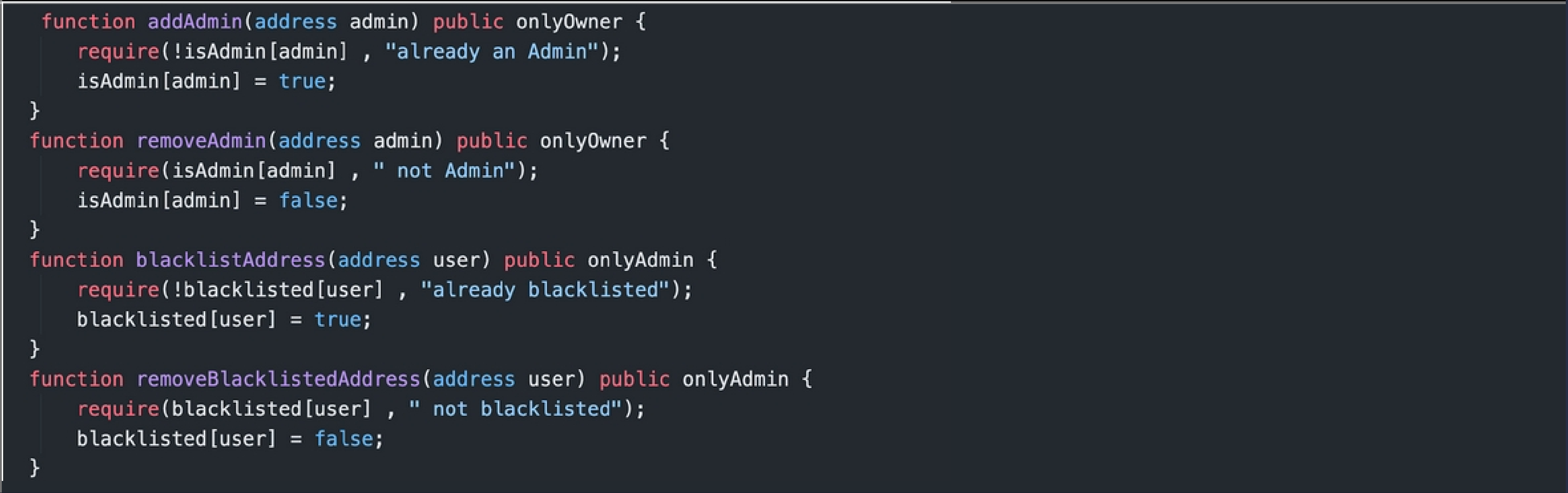
No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

A.1 Missing events for significant actions

Line	Function - addAdmin(), removeAdmin(), blacklistAddress(), removeBlacklistedAddress()
313, 317, 321, 325	

```
 function addAdmin(address admin) public onlyOwner {
     require(!isAdmin[admin] , "already an Admin");
     isAdmin[admin] = true;
}
function removeAdmin(address admin) public onlyOwner {
     require(isAdmin[admin] , " not Admin");
     isAdmin[admin] = false;
}
function blacklistAddress(address user) public onlyAdmin {
     require(!blacklisted[user] , "already blacklisted");
     blacklisted[user] = true;
}
function removeBlacklistedAddress(address user) public onlyAdmin {
     require(blacklisted[user] , " not blacklisted");
     blacklisted[user] = false;
}
```

Description

Whenever certain significant privileged actions are performed within the contract, it is recommended to emit an event about it. adding an Admin, removing an Admin, blacklisting an address, removing an address from the blacklist, etc are some significant actions hence it is recommended to emit an event for this action.

Remediation

Consider emitting an event whenever any of these four functions (addAdmin(), removeAdmin(), blacklistAddress(), removeBlacklistedAddress()) is invoked.

Status

Fixed in Version 1

Informational Issues

A.2 Multiple Solidity Pragma

Description

Contracts should be deployed with the same compiler version and flags that they have been tested 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. Additionally, It is better to use one Solidity compiler version across all contracts instead of different versions with different bugs and security checks.

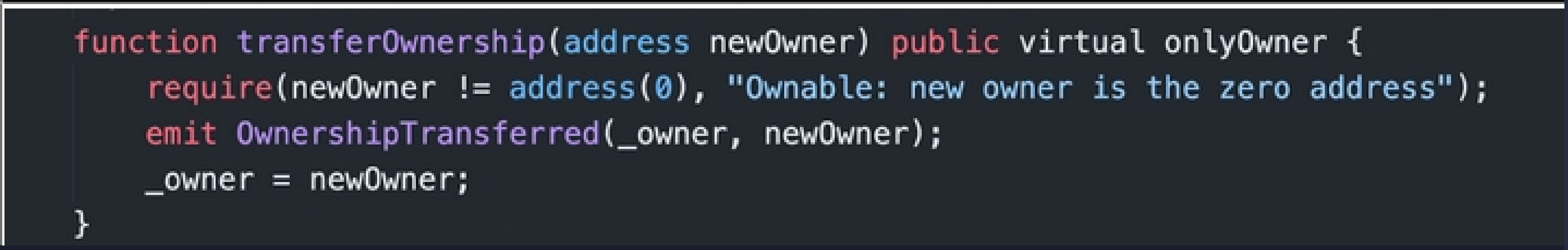
Remediation

Lock the pragma version by removing the ^ sign to lock the file onto a specific Solidity version. Moreover, consider using the same solidity compiler version throughout the code.

Status

Fixed in version 2

A.3 Ownership Transfer must be a two-step process.

Line	Function - transferOwnership()
181	

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
```

Description

The transferOwnership() function in contract allows the current admin to transfer his privileges to another address. However, inside transferOwnership() , the newOwner is directly stored into the storage, owner, after validating the newOwner is a non-zero address, which may not be enough.

As shown in the above code snippets, newOwner is only validated against the zero address. However, if the current admin enters a wrong address by mistake, he would never be able to take the management permissions back. Besides, if the newOwner is the same as the current admin address stored in _owner , it's a waste of gas.

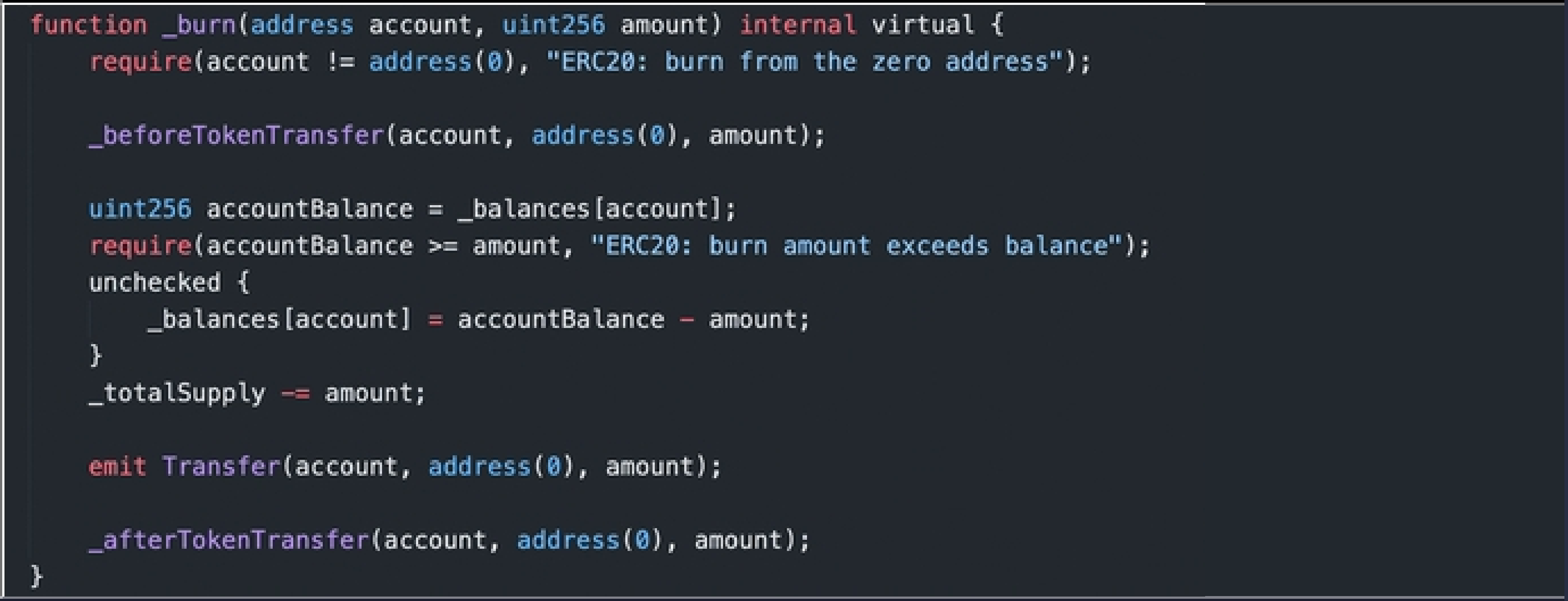
Remediation

It would be much safer if the transition is managed by implementing a two-step approach: _transferOwnership() and _updateOwnership() . Specifically, the _transferOwnership () function keeps the new address in the storage, _newOwner ,instead of modifying the _owner() directly. The updateOwnership() function checks whether _newOwner is msg.sender , which means _newOwner signs the transaction and verifies himself as the new owner. After that, _newOwner could be set into _owner.

Status

Fixed in version 1

A.4: Unused Code

Line	Function - _burn()
485	

```
function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

    _beforeTokenTransfer(account, address(0), amount);

    uint256 accountBalance = _balances[account];
    require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
    unchecked {
        _balances[account] = accountBalance - amount;
    }
    _totalSupply -= amount;

    emit Transfer(account, address(0), amount);

    _afterTokenTransfer(account, address(0), amount);
}
```

Description

In the contract, there an internal function named _burn(). This function is not called anywhere inside the contract. It is recommended to remove unused functions from the contract.

Remediation

It is recommended to review the logic of the contract and remove the unused functions.

Status

Fixed in version 1

A.5: Public functions that could be declared external in order to save gas.

Description

Whenever a function is not called internally, it is recommended to define them as external instead of public in order to save gas. For all the public functions, the input parameters are copied to memory automatically, and it costs gas. If your function is only called externally, then you should explicitly mark it as external. External function's parameters are not copied into memory but are read from calldata directly. This small optimization in your solidity code can save you a lot of gas when the function input parameters are huge.
Here is a list of functions that could be declared external:

- renounceOwnership() #172-175
- transferOwnership(address) #181-185
- name() #249-251
- symbol() #257-259
- decimals() #274-276
- totalSupply() #281-283
- balanceOf(address) #288-290
- transfer(address,uint256) #300-304
- addAdmin(address) #313-316
- removeAdmin(address) #317-320
- blacklistAddress(address) #321-324
- removeBlacklistedAddress(address)#325-328
- approve(address,uint256) #339-343
- transferFrom #361-371
- increaseAllowance(address,uint256) #385-389
- decreaseAllowance(address,uint256) #405-414

Remediation

Consider declaring these functions as external instead of public.

Status

Fixed in version 1

Functional Testing

A. Contract - BridgeToken.sol

- ✓ Should test renounceOwnership.
- ✓ Should test transferOwnership.
- ✓ Should call AddAdmin and verify access controls.
- ✓ Should call balcklistAddress and verify access controls.
- ✓ Should removeAdmin and removeBlacklisted.
- ✓ Should test approve, transferFrom.
- ✓ Only owner and admin must be able to blacklist an address.
- ✓ should increase and decrease allowance.
- ✓ BlackListed address must not be able to transact.

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 the Bridge Network Smart Contract. We performed our audit according to the procedure described above.

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

At the end,the Bridge Network team resolved all issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Bridge Network Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Bridge Network Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

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.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2022

For

Bridge

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com