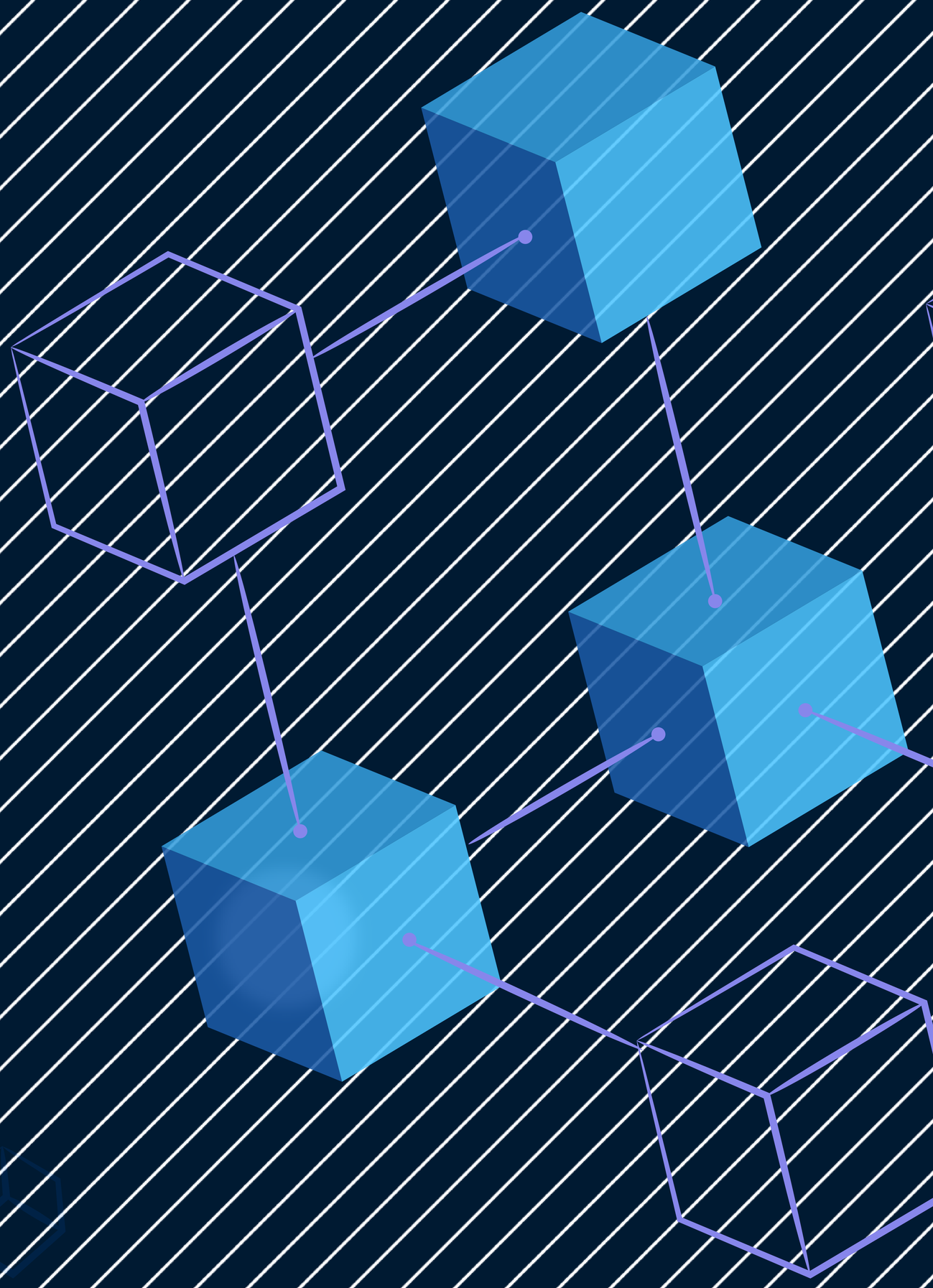
QuillAudits

Audit Report
October, 2021



For

ACKNOLEDGER

Contents

Scope of the Audit

The scope of this audit was to analyze and document the AcknoLedger smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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 level

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, Theo.

Issue Categories

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.

Risk-level	Description
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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	0	0
Closed	0	0	0	4

Introduction

During the period of **October 09, 2021, to October 14, 2021** - QuillAudits Team performed a security audit for **AcknoLedger** smart contracts.

The code for the audit was taken from following the official link: https://github.com/DefiWizard/acknoledger-contracts

Note	Date	Commit hash	Files
Version 1	October 9	1bcd31f92b8ce7cbcc113 a39e80de3481d5e7ad0	AcknoLedgerToken.sol
Version 2	October 14	1f559e27074c199f6b01 75a724eb4a145a1638d0	AcknoLedgerToken.sol

Issues Found

A. Contract – AcknoLedger

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

Informational issues

1. **Public function that could be declared external**

   Description

   The following public functions that are never called by the contract should be declared external to save gas:
   - setGovernance

   Remediation

   Use the external attribute for functions never called from the contract.

   Status: Fixed

   In version 2, the AcknoLedger team followed the recommended changes.

2. Missing Events for Significant Transactions

Line	Code

28-31	constructor() ERC20("AcknoLedgerToken", "ACK") {
	governance = msg.sender;
	_mint(governance, MAX_CAP);
	}

38-40	function setGovernance(address _governance) public onlyGovernance {
	governance = _governance;
	}

Description
The missing event makes it difficult to track off-chain decimal changes. An event should be emitted for significant transactions changing the following variables:
- governance

Remediation
We recommend emitting the appropriate events.

Status: Fixed
In version 2, the AcknoLedger team followed the recommended changes.

3. Ownable contract not required

Line	Code
16	contract AcknoLedgerToken is ERC20Permit, Ownable {

Description
The contract ‘Ownable’ is inherited by the AcknoLedgerToken contract. But on our testing we found out that no functionality of the Ownable contract is used inside the AcknoLedgerToken contract.

Remediation
We recommend that the Ownable contract should not be inherited, as this could help decrease the code size.

Status: Fixed
In version 2, the AcknoLedger team followed the recommended changes.

4. Missing zero address validation

Line	Code

38-40	function setGovernance(address _governance) public onlyGovernance {
	governance = _governance;
	}

49-57	function recoverToken(
	address token,
	address destination,
	uint256 amount
	) external onlyGovernance {
	require(token != destination, "Invalid address");
	require(IERC20(token).transfer(destination, amount), "Retrieve failed");
	emit RecoverToken(token, destination, amount);
	}

Description
When setting the governance address, it should be checked for zero address. Otherwise, they may lose the ownership of the contract if the function is called with address(0) as the parameter.
Similarly, the destination address should be checked for zero address. Otherwise, tokens sent to the zero address may be burnt forever.

Remediation
Use a require statement to check for zero addresses.

Status: Fixed
In version 2, the AcknoLedger team followed the recommended changes.

Functional test

Function Names	Testing results
name()	Passed
symbol()	Passed
decimals()	Passed
totalSupply()	Passed
balanceOf()	Passed
transfer()	Passed
allowance()	Passed
approve()	Passed
transferFrom()	Passed
increaseAllowance()	Passed
decreaseAllowance()	Passed
setGovernance()	Passed
recoverToken()	Passed
permit()	Passed
nonces()	Passed

Automated Tests

Slither

```
AcknoLedgerToken.setGovernance(address) (AcknoLedgerToken.sol#38-40) should emit an event for:
        - governance = _governance (AcknoLedgerToken.sol#39)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

AcknoLedgerToken.setGovernance(address)._governance (AcknoLedgerToken.sol#38) lacks a zero-check on :
                - governance = _governance (AcknoLedgerToken.sol#39)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in AcknoLedgerToken.recoverToken(address,address,uint256) (AcknoLedgerToken.sol#49-57):
        External calls:
        - require(bool,string)(IERC20(token).transfer(destination,amount),Retrieve failed) (AcknoLedgerToken.sol#55)
        Event emitted after the call(s):
        - RecoverToken(token,destination,amount) (AcknoLedgerToken.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (ERC20Permit.sol#40-61) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,Permit: expired deadline) (ERC20Permit.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.isContract(address) (@openzeppelin\contracts\utils\Address.sol#26-36) uses assembly
        - INLINE ASM (@openzeppelin\contracts\utils\Address.sol#32-34)
Address.verifyCallResult(bool,bytes,string) (@openzeppelin\contracts\utils\Address.sol#195-215) uses assembly
        - INLINE ASM (@openzeppelin\contracts\utils\Address.sol#207-210)
ERC20Permit.constructor() (ERC20Permit.sol#19-34) uses assembly
        - INLINE ASM (ERC20Permit.sol#21-23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
        - Version used: ['0.8.0', '^0.8.0']
        - 0.8.0 (AcknoLedgerToken.sol#3)
        - ^0.8.0 (@openzeppelin\contracts\utils\Address.sol#3)
        - ^0.8.0 (@openzeppelin\contracts\utils\Context.sol#3)
        - ^0.8.0 (@openzeppelin\contracts\utils\Counters.sol#3)
        - ^0.8.0 (@openzeppelin\contracts\token\ERC20\ERC20.sol#3)
        - 0.8.0 (ERC20Permit.sol#3)
        - ^0.8.0 (@openzeppelin\contracts\token\ERC20\IERC20.sol#3)
        - ^0.8.0 (@openzeppelin\contracts\token\ERC20\extensions\IERC20Metadata.sol#3)
        - 0.8.0 (IERC2612Permit.sol#3)
        - ^0.8.0 (@openzeppelin\contracts\access\Ownable.sol#3)
        - ^0.8.0 (@openzeppelin\contracts\token\ERC20\utils\SafeERC20.sol#3)
        - ^0.8.0 (@openzeppelin\contracts\utils\math\SafeMath.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version0.8.0 (AcknoLedgerToken.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\utils\Address.sol#3) necessitates a version too recent to be trusted. Consider deploying
with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\utils\Context.sol#3) necessitates a version too recent to be trusted. Consider deploying
with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\utils\Counters.sol#3) necessitates a version too recent to be trusted. Consider deploying
 with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\ERC20.sol#3) necessitates a version too recent to be trusted. Consider deploy
ing with 0.6.12/0.7.6
Pragma version0.8.0 (ERC20Permit.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
```

```
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\IERC20.sol#3) necessitates a version too recent to be trusted. Consider deplo
ying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\extensions\IERC20Metadata.sol#3) necessitates a version too recent to be trus
ted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.0 (IERC2612Permit.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\access\Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying
 with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\utils\SafeERC20.sol#3) necessitates a version too recent to be trusted. Consi
der deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\utils\math\SafeMath.sol#3) necessitates a version too recent to be trusted. Consider depl
oying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (@openzeppelin\contracts\utils\Address.sol#54-59):
        - (success) = recipient.call{value: amount}() (@openzeppelin\contracts\utils\Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (@openzeppelin\contracts\utils\Address.sol#122-133):
        - (success,returndata) = target.call{value: value}(data) (@openzeppelin\contracts\utils\Address.sol#131)
Low level call in Address.functionStaticCall(address,bytes,string) (@openzeppelin\contracts\utils\Address.sol#151-160):
        - (success,returndata) = target.staticcall(data) (@openzeppelin\contracts\utils\Address.sol#158)
Low level call in Address.functionDelegateCall(address,bytes,string) (@openzeppelin\contracts\utils\Address.sol#178-187):
        - (success,returndata) = target.delegatecall(data) (@openzeppelin\contracts\utils\Address.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter AcknoLedgerToken.setGovernance(address)._governance (AcknoLedgerToken.sol#38) is not in mixedCase
Variable ERC20Permit.DOMAIN_SEPARATOR (ERC20Permit.sol#17) is not in mixedCase

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

setGovernance(address) should be declared external:
        - AcknoLedgerToken.setGovernance(address) (AcknoLedgerToken.sol#38-40)
symbol() should be declared external:
        - ERC20.symbol() (@openzeppelin\contracts\token\ERC20\ERC20.sol#69-71)
decimals() should be declared external:
        - ERC20.decimals() (@openzeppelin\contracts\token\ERC20\ERC20.sol#86-88)
totalSupply() should be declared external:
        - ERC20.totalSupply() (@openzeppelin\contracts\token\ERC20\ERC20.sol#93-95)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (@openzeppelin\contracts\token\ERC20\ERC20.sol#100-102)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (@openzeppelin\contracts\token\ERC20\ERC20.sol#112-115)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (@openzeppelin\contracts\token\ERC20\ERC20.sol#120-122)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (@openzeppelin\contracts\token\ERC20\ERC20.sol#131-134)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (@openzeppelin\contracts\token\ERC20\ERC20.sol#149-163)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (@openzeppelin\contracts\token\ERC20\ERC20.sol#177-180)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (@openzeppelin\contracts\token\ERC20\ERC20.sol#196-204)
permit(address,address,uint256,uint256,uint8,bytes32,bytes32) should be declared external:
        - ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (ERC20Permit.sol#40-61)

nonces(address) should be declared external:
        - ERC20Permit.nonces(address) (ERC20Permit.sol#66-68)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (@openzeppelin\contracts\access\Ownable.sol#53-55)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (@openzeppelin\contracts\access\Ownable.sol#61-64)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
. analyzed (12 contracts with 75 detectors), 42 result(s) found
```

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or endorsement of the **AcknoLedger 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 **AcknoLedger** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
October, 2021

For

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com