QuillAudits

Audit Report
February, 2022

For

Contents

Scope of the Audit

The scope of this audit was to analyze and document the Poollotto Token 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	1	0
Closed	0	0	0	0

Introduction

During the period of **February 15, 2022 to February 17, 2022** - QuillAudits Team performed a security audit for the Poollotto smart contract.

The code for the Poollotto contract was obtained from:
https://bscscan.com/address/0x631C2f0EdABaC799f07550aEE4fF0Bf7fd35212B#code

The contract was deployed and tested on Ropsten and you can find it here:
Poollotto: 0xD4c196069F06f27FB045b7964243c48870919e5a

Issues Found – Code Review / Manual Testing

A.Contract - HarmonyLauncher.sol

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

1. **Permit function**

   The permit function was not able to recognize the signer's address using the v, r and s provided. The version being used here is a draft version of the OpenZeppelin contract (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/draft-ERC20Permit.sol) and not a standard. A more accurate permit function can be found here:

   https://github.com/soliditylabs/ERC20-Permit/blob/main/contracts/ERC20Permit.sol

   Status: Acknowledged

Informational issues

No issues were found.

Functional Tests

Function Names	Testing results
transfer	Passed
transferFrom	Passed
burn	Passed
burnFrom	Passed
permit	Failed
approve	Passed
increaseAllowance	Passed
decreaseAllowance	Passed

Functionality Tests Performed

- Users should be able to transfer tokens not more than their balance. PASS
- Users should not be able to transfer if any of the conditions in the transactionThrottler are not met. PASS
- approve. PASS
- Users should be able to transferFrom tokens not more than their approval and also not more than the owner's(token owner) balance. PASS
- Users should not be able to transferFrom if any of the conditions in the transactionThrottler are not met. PASS
- Users should be able to burn tokens not more than their balance. PASS
- Users should be able to burnFrom tokens not more than their approval. PASS
- Users should be able to increaseAllowance. PASS
- Users should be able to decreaseAllowance. PASS
- Users should be able to call permit to approve other users using a signed message. FAIL

Automated Tests

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum, Hedera, Quorum, Vechain, Roostock, Tron and other EVM-compatible blockchains. It uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities. Mythril raised the following concerns:

```
==== Dependence on predictable environment variable ====
SWC ID: 116
Severity: Low
Contract: Poollotto
Function name: permit(address,address,uint256,uint256,uint8,bytes32,bytes32) or permit(address,address,uint256,uint256,uint8,bytes32,bytes32)
or permit(address,address,uint256,uint256,uint8,bytes32,bytes32)
PC address: 2250
Estimated Gas Usage: 1605 - 1700
A control flow decision is made based on The block.timestamp environment variable.
The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslim
it, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of ea
rlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a cert
ain level of trust into miners.
--------------------
In file: PL.sol:882

require(block.timestamp <= deadline, "ERC20Permit: expired deadline")

--------------------
Initial State:

Account: [CREATOR], balance: 0x800008, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x2, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0
Caller: [ATTACKER], function: permit(address,address,uint256,uint256,uint8,bytes32,bytes32), txdata: 0xd505accf0000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000, value:
0x0
```

Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to these in real-time.
We performed analysis using the contract Library on the Ropsten address of the Poollotto contract used during manual testing:

Poollotto: 0xD4c196069F06f27FB045b7964243c48870919e5a

Slither

Slither is a Solidity static analysis framework written in Python 3. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

```
ERC20Permit.constructor(string).name (PL.sol#874) shadows:
        - ERC20.name() (PL.sol#201-203) (function)
        - IERC20Metadata.name() (PL.sol#101) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (PL.sol#880-901) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (PL.sol#882)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

ECDSA.recover(bytes32,bytes) (PL.sol#608-641) uses assembly
        - INLINE ASM (PL.sol#621-625)
        - INLINE ASM (PL.sol#630-635)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Context._msgData() (PL.sol#135-138) is never used and should be removed
Counters.decrement(Counters.Counter) (PL.sol#831-837) is never used and should be removed
ECDSA.recover(bytes32,bytes) (PL.sol#608-641) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes32) (PL.sol#675-679) is never used and should be removed
ERC20Capped._mint(address,uint256) (PL.sol#521-524) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Function IERC20Permit.DOMAIN_SEPARATOR() (PL.sol#578) is not in mixedCase
Variable EIP712._CACHED_DOMAIN_SEPARATOR (PL.sol#725) is not in mixedCase
Variable EIP712._CACHED_CHAIN_ID (PL.sol#726) is not in mixedCase
Variable EIP712._HASHED_NAME (PL.sol#728) is not in mixedCase
Variable EIP712._HASHED_VERSION (PL.sol#729) is not in mixedCase
Variable EIP712._TYPE_HASH (PL.sol#730) is not in mixedCase
Function ERC20Permit.DOMAIN_SEPARATOR() (PL.sol#914-916) is not in mixedCase
Variable ERC20Permit._PERMIT_TYPEHASH (PL.sol#867) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (PL.sol#136)" inContext (PL.sol#130-139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

name() should be declared external:
        - ERC20.name() (PL.sol#201-203)
symbol() should be declared external:
        - ERC20.symbol() (PL.sol#209-211)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (PL.sol#240-242)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (PL.sol#252-255)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (PL.sol#271-274)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (PL.sol#289-297)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (PL.sol#311-314)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (PL.sol#330-336)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (PL.sol#466-468)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (PL.sol#481-486)
permit(address,address,uint256,uint256,uint8,bytes32,bytes32) should be declared external:
        - ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (PL.sol#880-901)
nonces(address) should be declared external:
        - ERC20Permit.nonces(address) (PL.sol#906-908)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
PL.sol analyzed (12 contracts with 77 detectors), 29 result(s) found
```

Results

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 Poollotto. We performed our audit according to the procedure described above.The audit showed one Low severity issue,which has been Acknowledged by the Poollotto Team.

Disclaimer

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

Audit Report
February, 2022

For

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com