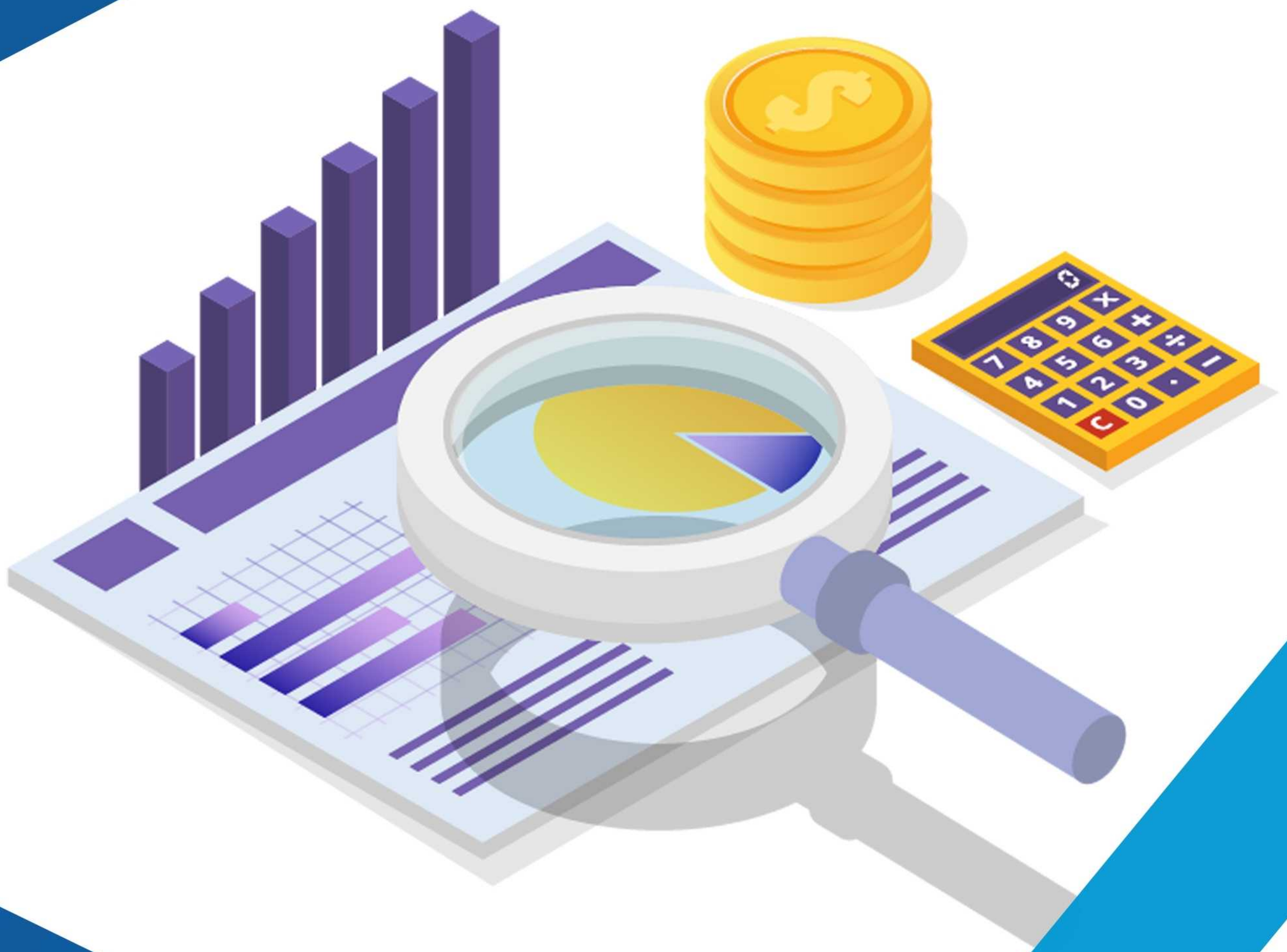
QuillAudits



Audit Report
July, 2021

Contents

Scope of Audit

The scope of this audit was to analyze and document the TIME 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 analyzed 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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability 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 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 possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with 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 to be fixed before moving to a live environment.

Medium level 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 level severity issues

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 four issues which 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	1
Acknowledged	0	1	0	1
Closed	0	0	0	0

Introduction

During the period of **July 12, 2021 to July 15, 2021** - QuillAudits Team performed a security audit for TIME smart contracts.

The code for the audit was taken from following the official link:

ETH:
https://etherscan.io/address/0x8551fe601a77b80572d69278ef0cf046356093da#code

BSC:
https://bscscan.com/address/0x33bba52ed721ed5b2d1fb4588f88f2f6522d6e11#code

Issues Found – Code Review / Manual Testing

High severity issues

No issues were found.

Medium severity issues

1. **Centralization Risks**

   Description

   The role owner has the authority to
   update the critical settings
   manage the list containing contracts excluding from setBaseCurrency, proofOfLapse, removeTimeZone and addOrUpdateTimeZone.

   Remediation

   We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:

   1. with reasonable latency for community awareness on privileged operations;
   2. Multisig with community-voted 3rd-party independent co-signers;
   3. DAO or Governance module increasing transparency and community involvement;

   Status: Acknowledged

Low level severity issues

No issues were found.

Informational

2. block.timestamp may not be reliable

Description
The Time contract uses the block.timestamp as part of the calculations and time checks.

Nevertheless, timestamps can be slightly altered by miners to favor them in contracts that have logics that depend strongly on them. Consider taking into account this issue and warning the users that such a scenario could happen.

Status: Acknowledged

3. Functions with similar name

Description
The Time contract includes a function with exactly a similar name. Since every function executes with different parameters, it is considered a better practice to avoid similar names for the proofOfLapse function.

Status: Open

Functional test

Function Names	Testing results
addOrUpdateTimeZone	Passed
approve	Passed
burn	Passed
buyForBase	Passed
cleanTimeZones	Passed
cleanVotingProportion	Passed
decreaseAllowance	Passed
increaseAllowance	Passed
initialize	Passed
mint	Passed
proofOfLapse	Passed
proofOfLapse	Passed
Reimburse	Passed
removeTimeZone	Passed
renounceOwnership	Passed
sellForBase	Passed
setBaseCurrency	Passed
setLiquidity	Passed
transfer	Passed
transferFrom	Passed
transferOwnership	Passed
unvote	Passed
vote	Passed

Automated Testing

Slither

```
INFO:Detectors:
Time._computePriceInfo(uint256) (final_tmp.sol#1094-1123) uses a weak PRNG: "secondInDay
= timeStamp % _secondsInDay() (final_tmp.sol#1104)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
Time.buyForBase(uint256) (final_tmp.sol#1128-1135) ignores return value by
IERC20(_baseCurrency).transferFrom(msg.sender,_liquidity,baseAmount) (final_tmp.sol#1131)
Time.sellForBase(uint256) (final_tmp.sol#1140-1148) ignores return value by
IERC20(_baseCurrency).transferFrom(_liquidity,msg.sender,baseAmount) (final_tmp.sol#1144)
Time.reimburse() (final_tmp.sol#1153-1159) ignores return value by
IERC20(_baseCurrency).transferFrom(_liquidity,msg.sender,_expense) (final_tmp.sol#1155)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-
transfer
INFO:Detectors:
Time._computePriceInfo(uint256) (final_tmp.sol#1094-1123) uses a dangerous strict equality:
    - secondInDay == 0 (final_tmp.sol#1111)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-
equalities
INFO:Detectors:
Reentrancy in Time.reimburse() (final_tmp.sol#1153-1159):
    External calls:
    - IERC20(_baseCurrency).transferFrom(_liquidity,msg.sender,_expense)
(final_tmp.sol#1155)
    State variables written after the call(s):
    - _expense = 0 (final_tmp.sol#1157)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-1
INFO:Detectors:
Time.initialize(string,string,uint8,uint256,bool,address,address,address).name
(final_tmp.sol#884) shadows:
    - ERC20TokenImplementation.name() (final_tmp.sol#491-493) (function)
    - IERC20.name() (final_tmp.sol#103) (function)
Time.initialize(string,string,uint8,uint256,bool,address,address,address).symbol
(final_tmp.sol#884) shadows:
    - ERC20TokenImplementation.symbol() (final_tmp.sol#484-486) (function)
    - IERC20.symbol() (final_tmp.sol#98) (function)
Time.initialize(string,string,uint8,uint256,bool,address,address,address).decimals
(final_tmp.sol#884) shadows:
    - ERC20TokenImplementation.decimals() (final_tmp.sol#477-479) (function)
    - IERC20.decimals() (final_tmp.sol#93) (function)
```

```
Time.initialize(string,string,uint8,uint256,bool,address,address,address).mintable
(final_tmp.sol#884) shadows:
    - ERC20TokenImplementation.mintable() (final_tmp.sol#463-465) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-
shadowing
INFO:Detectors:
Reentrancy in Time.buyForBase(uint256) (final_tmp.sol#1128-1135):
    External calls:
    - IERC20(_baseCurrency).transferFrom(msg.sender,_liquidity,baseAmount)
(final_tmp.sol#1131)
    State variables written after the call(s):
    - _mint(msg.sender,timeAmount) (final_tmp.sol#1132)
        - _balances[account] = _balances[account].add(amount) (final_tmp.sol#654)
    - _mint(msg.sender,timeAmount) (final_tmp.sol#1132)
        - _totalSupply = _totalSupply.add(amount) (final_tmp.sol#653)
Reentrancy in Time.sellForBase(uint256) (final_tmp.sol#1140-1148):
    External calls:
    - IERC20(_baseCurrency).transferFrom(_liquidity,msg.sender,baseAmount)
(final_tmp.sol#1144)
    State variables written after the call(s):
    - _burn(msg.sender,timeAmount) (final_tmp.sol#1145)
        - _balances[account] = _balances[account].sub(amount,ERC20: burn amount exceeds
balance) (final_tmp.sol#672)
    - _burn(msg.sender,timeAmount) (final_tmp.sol#1145)
        - _totalSupply = _totalSupply.sub(amount) (final_tmp.sol#673)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-2
INFO:Detectors:
Reentrancy in Time.buyForBase(uint256) (final_tmp.sol#1128-1135):
    External calls:
    - IERC20(_baseCurrency).transferFrom(msg.sender,_liquidity,baseAmount)
(final_tmp.sol#1131)
    Event emitted after the call(s):
    - TimeBought(timeAmount,baseAmount,msg.sender) (final_tmp.sol#1133)
    - Transfer(address(0),account,amount) (final_tmp.sol#655)
        - _mint(msg.sender,timeAmount) (final_tmp.sol#1132)
Reentrancy in Time.reimburse() (final_tmp.sol#1153-1159):
    External calls:
    - IERC20(_baseCurrency).transferFrom(_liquidity,msg.sender,_expense) (final_tmp.sol#1155)
    Event emitted after the call(s):
    - Reimbursed(_expense) (final_tmp.sol#1156)
Reentrancy in Time.sellForBase(uint256) (final_tmp.sol#1140-1148):
    External calls:
    - IERC20(_baseCurrency).transferFrom(_liquidity,msg.sender,baseAmount)
(final_tmp.sol#1144)
    Event emitted after the call(s):
```

```
    - TimeSold(timeAmount,baseAmount,msg.sender) (final_tmp.sol#1146)
    - Transfer(account,address(0),amount) (final_tmp.sol#674)
        - _burn(msg.sender,timeAmount) (final_tmp.sol#1145)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Time._computePriceInfo(uint256) (final_tmp.sol#1094-1123) uses timestamp for comparisons
    Dangerous comparisons:
    - timeStamp <= _minTimeStamp (final_tmp.sol#1095)
    - _maxTimeStamp <= timeStamp (final_tmp.sol#1099)
    - secondInDay == 0 (final_tmp.sol#1111)
Time.buyForBase(uint256) (final_tmp.sol#1128-1135) uses timestamp for comparisons
    Dangerous comparisons:
    - require(bool,string)(timeAmount * currentEndPrice() > 10000000000000000,Provided timeAmount is too small!) (final_tmp.sol#1129)
Time.sellForBase(uint256) (final_tmp.sol#1140-1148) uses timestamp for comparisons
    Dangerous comparisons:
    - require(bool,string)(timeAmount * currentPrice() > 10000000000000000,Provided timeAmount is too small!) (final_tmp.sol#1141)
    - require(bool,string)(maxTimeSellAmount() > timeAmount,Time amount to sell should be lower than maxTimeSellAmount()!) (final_tmp.sol#1142)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Initializable._isConstructor() (final_tmp.sol#397-408) uses assembly
    - INLINE ASM (final_tmp.sol#406)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Context._msgData() (final_tmp.sol#190-194) is never used and should be removed
ERC20TokenImplementation._burnFrom(address,uint256) (final_tmp.sol#704-707) is never used and should be removed
SafeMath.div(uint256,uint256) (final_tmp.sol#295-297) is never used and should be removed
SafeMath.div(uint256,uint256,string) (final_tmp.sol#311-317) is never used and should be removed
SafeMath.mod(uint256,uint256) (final_tmp.sol#331-333) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (final_tmp.sol#347-350) is never used and should be removed
SafeMath.mul(uint256,uint256) (final_tmp.sol#269-281) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.1 (final_tmp.sol#82) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.1 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

INFO:Detectors:
Variable ERC20TokenImplementation._balances (final_tmp.sol#414) is not in mixedCase
Variable ERC20TokenImplementation._allowances (final_tmp.sol#415) is not in mixedCase
Variable ERC20TokenImplementation._totalSupply (final_tmp.sol#416) is not in mixedCase
Variable ERC20TokenImplementation._name (final_tmp.sol#417) is not in mixedCase
Variable ERC20TokenImplementation._symbol (final_tmp.sol#418) is not in mixedCase
Variable ERC20TokenImplementation._decimals (final_tmp.sol#419) is not in mixedCase
Variable ERC20TokenImplementation._owner (final_tmp.sol#421) is not in mixedCase
Variable ERC20TokenImplementation._mintable (final_tmp.sol#425) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (final_tmp.sol#192)" inContext (final_tmp.sol#185-195)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
renounceOwnership() should be declared external:
    - ERC20TokenImplementation.renounceOwnership() (final_tmp.sol#445-448)
transferOwnership(address) should be declared external:
    - ERC20TokenImplementation.transferOwnership(address) (final_tmp.sol#454-458)
increaseAllowance(address,uint256) should be declared external:
    - ERC20TokenImplementation.increaseAllowance(address,uint256) (final_tmp.sol#571-574)
decreaseAllowance(address,uint256) should be declared external:
    - ERC20TokenImplementation.decreaseAllowance(address,uint256) (final_tmp.sol#590-593)
mint(uint256) should be declared external:
    - ERC20TokenImplementation.mint(uint256) (final_tmp.sol#604-608)
burn(uint256) should be declared external:
    - ERC20TokenImplementation.burn(uint256) (final_tmp.sol#613-616)
currentStartPrice() should be declared external:
    - Time.currentStartPrice() (final_tmp.sol#1042-1045)
currentStartTimeStamp() should be declared external:
    - Time.currentStartTimeStamp() (final_tmp.sol#1059-1062)
currentEndTimeStamp() should be declared external:
    - Time.currentEndTimeStamp() (final_tmp.sol#1067-1070)
currentPriceInfo() should be declared external:
    - Time.currentPriceInfo() (final_tmp.sol#1075-1078)
historicalPriceInfo(uint256) should be declared external:
    - Time.historicalPriceInfo(uint256) (final_tmp.sol#1090-1092)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

Mythril

```
==== Dependence on predictable environment variable ====
SWC ID: 116
Severity: Low
Contract: 0xc7Fd594512e3064745A7cb52b7eD29c19ce388a8
Function name: currentPriceInfo()
PC address: 9264
Estimated Gas Usage: 2228 - 5628
A control flow decision is made based on a predictable variable.
The block.timestamp environment variable is used in to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables for random number generation or to make critical control flow decisions.
--------------------
Initial State:

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

Transaction Sequence:

Caller: [CREATOR], data: [CONTRACT CREATION], value: 0x0
```

THEO

```
theo --rpc-http http://127.0.0.1:8545
The account's private key (input hidden)
>
Contract to interact with
> 0xc7Fd594512e3064745A7cb52b7eD29c19ce388a8
Scanning for exploits in contract: 0xc7Fd594512e3064745A7cb52b7eD29c19ce388a8
Connecting to HTTP: http://127.0.0.1:8545.
No exploits found. You're going to need to load some exploits.

Tools available in the console:
- `exploits` is an array of loaded exploits found by Mythril or read from a file
- `w3` an initialized instance of web3py for the provided HTTP RPC endpoint
- `dump()` writing a json representation of an object to a local file

Check the readme for more info:
https://github.com/cleanunicorn/theo

Theo version v0.8.2.
```

SOLHINT LINTER

```
    final_tmp.sol
    82:1   error    Compiler version 0.8.1 does not satisfy the ^0.5.8 semver requirement
compiler-version
   278:9   warning  Error message for require is too long
reason-string
   381:9   warning  Error message for require is too long
reason-string
   427:5   warning  Explicitly mark visibility in function (Set ignoreConstructors to true if using
solidity >=0.7.0)  func-visibility
   427:19  warning  Code contains empty blocks
no-empty-blocks
   455:9   warning  Error message for require is too long
reason-string
   633:9   warning  Error message for require is too long
reason-string
   634:9   warning  Error message for require is too long
reason-string
   670:9   warning  Error message for require is too long
reason-string
   691:9   warning  Error message for require is too long
reason-string
   692:9   warning  Error message for require is too long
reason-string
   754:9   warning  Error message for require is too long
reason-string
   756:9   warning  Error message for require is too long
reason-string
   810:9   warning  Error message for require is too long
reason-string
   812:9   warning  Error message for require is too long
reason-string
   873:5   warning  Explicitly mark visibility in function (Set ignoreConstructors to true if using
solidity >=0.7.0)  func-visibility
   886:9   warning  Error message for require is too long
reason-string
   887:9   warning  Error message for require is too long
reason-string
   919:9   warning  Error message for require is too long
reason-string
   927:9   warning  Error message for require is too long
reason-string
   928:9   warning  Error message for require is too long
reason-string
```

```
  929:9   warning  Error message for require is too long
reason-string
  935:9   warning  Error message for require is too long
reason-string
  984:9   warning  Error message for require is too long
reason-string
 1002:9   warning  Error message for require is too long
reason-string
 1035:36  warning  Avoid to make time-based decisions in your business logic
not-rely-on-time
 1043:36  warning  Avoid to make time-based decisions in your business logic
not-rely-on-time
 1052:36  warning  Avoid to make time-based decisions in your business logic
not-rely-on-time
 1060:36  warning  Avoid to make time-based decisions in your business logic
not-rely-on-time
 1068:36  warning  Avoid to make time-based decisions in your business logic
not-rely-on-time
 1076:36  warning  Avoid to make time-based decisions in your business logic
not-rely-on-time
 1129:9   warning  Error message for require is too long
reason-string
 1141:9   warning  Error message for require is too long
reason-string
 1142:9   warning  Error message for require is too long
reason-string
 1154:9   warning  Error message for require is too long
reason-string
 1169:9   warning  Error message for require is too long
reason-string
 1179:9   warning  Error message for require is too long
reason-string
```

Disclaimer

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

Closing Summary

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

The audit showed some Medium and Informational severity issues. Some changes were proposed to follow the best practices and reduce the potential attacks. As a result, the Auditee was cognizant of and acknowledged the issues.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com