Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Theta Token
Website: thetatoken.org
Platform: Ethereum
Language: Solidity
Date: May 13th, 2024

Table of contents

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Theta token smart contract from thetatoken.org was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 13th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- This Solidity code is an ERC20-compliant token contract called "ThetaToken" with additional functionalities. Here's a breakdown of its main components:
  - **SafeMath Library:** This library provides arithmetic operations with safety checks to prevent overflows and underflows.
  - **ERC20 Interface:** The ERC20 contract defines the standard ERC20 interface with functions like totalSupply, balanceOf, transfer, transferFrom, approve, and allowance.
  - **StandardToken Contract:** This contract implements the ERC20 interface and manages token balances and allowances using mappings. It also includes functions for transferring tokens (transfer and transferFrom), approving token transfers (approve), and checking allowances (allowance).
  - **Controlled Contract:** This contract defines a mechanism for controlling the contract. It allows the contract owner to change the controller address.
  - **ThetaToken Contract:** This is the main token contract that inherits from StandardToken and Controlled. It adds functionality specific to ThetaToken, such as:
    - Restricting token transfers until a certain unlock time (unlockTime).
    - Allowing pre-circulation of tokens for specific addresses before the unlock time.
    - Minting tokens, which increases the total supply and assigns new tokens to a specified owner address.

- Changing the unlock time.
        - Modifiers to check if token transfers are allowed (can_transfer).
- Overall, this contract allows for the creation of Theta tokens with additional features for controlling token transfers, minting new tokens, and managing pre-circulation before a specified unlock time.

Audit scope

Name	Code Review and Security Analysis Report for Theta Token Smart Contract
Platform	Ethereum
File	ThetaToken.sol
Smart Contract Code	0x3883f5e181fccaf8410fa61e12b59bad963fb645
Audit Date	May 13th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: Theta Token ● Symbol: THETA ● Decimals: 18 ● Total Supply: 1 billion	**YES, This is valid.**
Controller of the owner control: ● Mint the new tokens. ● Update Pre Circulation address. ● Update Unlock Time. ● Current Controller owner can set a new Controller address.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured"**. Also, these contracts contain owner control, which does not make them fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 3 low, and 6 very low-level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Moderated
	High consumption 'for/while' loop	Passed
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Moderated
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: PASSED

Business Risk Analysis

Category	Result
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	No
● Cannot Sell	No
● Max Tax	0%
● Modify Tax	Not Detected
● Fee Check	No
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	No
● Pause Transfer?	No
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	Not Detected
● Blacklist Check	No
● Can Mint?	Yes
● Is it a Proxy?	Not Detected
● Can Take Ownership?	Not Detected
● Hidden Owner?	Not Detected
● Self Destruction?	Not Detected
● Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Theta Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Theta Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are not well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Theta Token smart contract code in the form of an [Etherscan](#) web link.

As mentioned above, code parts are not well commented on. but the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well-known industry standard open source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	ThetaToken	write	can transfer	No Issue
3	transfer	write	Error message for require condition	Refer Audit Findings
4	transferFrom	write	Error message for require condition, Gas optimization	Refer Audit Findings
5	mint	write	Gas optimization, Centralized Risk, Controller can mint unlimited tokens:	Refer Audit Findings
6	allowPrecirculation	write	access only controller	No Issue
7	disallowPrecirculation	write	access only controller	No Issue
8	isPrecirculationAllowed	write	Passed	No Issue
9	changeUnlockTime	write	Critical operation lacks event log	Refer Audit Findings
10	getUnlockTime	write	Passed	No Issue
11	can_transfer	modifier	Error message for require condition	Refer Audit Findings
12	Controlled	write	Passed	No Issue
13	changeController	write	Critical operation lacks event log, Missing zero address validation	Refer Audit Findings
14	getController	write	Passed	No Issue
15	only_controller	modifier	Error message for require condition	Refer Audit Findings
16	totalSupply	write	Passed	No Issue
17	balanceOf	write	Passed	No Issue
18	transfer	write	Passed	No Issue
19	transferFrom	write	Passed	No Issue
20	approve	write	Error message for revert function, Gas optimization	Refer Audit Findings
21	allowance	write	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

(1) Critical operation lacks event log:

```solidity
function changeUnlockTime(uint _unlockTime) only_controller public {
        unlockTime = _unlockTime;
    }
function changeController(address _newController) public
only_controller {
        controller = _newController;
    }
```

Missing event log for:

- changeUnlockTime()
- change controller()

Resolution: Write an event log for listed events.

(2) Missing zero address validation:

```solidity
function changeController(address _newController) public
only_controller {
        controller = _newController;
    }
```

Detects missing zero address validation.

The changeController() without specifying the _newController, the Owner may lose ownership of the contract.

Resolution: We suggest first checking that the address is not zero.

(3) Controller can mint unlimited tokens:

```solidity
    function mint(address _owner, uint _amount) external
 only_controller returns (bool) {
        totalSupply = totalSupply.add(_amount);
        balances[_owner] = balances[_owner].add(_amount);

        Transfer(0, _owner, _amount);
        return true;
    }
```

There is no limit for minting tokens. Thus the controller can mint unlimited tokens to any account.

Resolution: There should be a limit for minting or need to confirm, if it is a part of the plan then disregard this issue.

Very Low / Informational / Best practices:

(1) Use the latest solidity version:

```solidity
pragma solidity ^0.4.18;
```

Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use versions greater than 0.8.7.

(2) Error message for revert function:

```solidity
function approve(address _spender, uint _value) public returns (bool
success) {
        //
```

```
https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
    if ((_value != 0) && (allowed[msg.sender][_spender] != 0)) {
        revert();
    }
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
}
```

Detects missing error messages in the approve() function.

Resolution: We suggest writing a proper message in revert(), Otherwise users can't identify the actual problem of why the transaction reverted.

(3) Error message for the required condition:

Detects missing error message in only_controller(), can_transfer(), transfer(), transferFrom() function.

Resolution: We suggest writing a proper message in require(), Otherwise users can't identify the actual problem of why the transaction reverted.

(4) Gas optimization:

The public functions that are never called by the contract could be declared external.

- approve
- transferFrom
- mint

Resolution: We suggest declaring these all functions external for better Gas optimization.

(5) Centralized Risk:

```
function mint(address _owner, uint _amount) external
only_controller returns (bool) {
        totalSupply = totalSupply.add(_amount);
        balances[_owner] = balances[_owner].add(_amount);


        Transfer(0, _owner, _amount);
```

```
        return true;
    }
```

Only only_controller can call the mint function.

Resolution: To make the smart contract 100% decentralized, we suggest renouncing ownership of the smart contract once its function is completed.

(6) Spelling mistakes:

Spelling mistake in declared variable comments line number > 150, // tokens can be transferred amoung holders only after unlockTime.

"amoung" word should be **"among".**

Resolution: We suggest correcting spelling mistakes.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble. The following are Admin functions:

ThetaToken.sol

- mint: The mint tokens by the controller of the owner.
- allowPrecirculation: Allow Pre Circulation address can be set by the controller of the owner.
- disallowPrecirculation: Disallow Pre Circulation address can be set by the controller of the owner.
- changeUnlockTime: Unlock Time can be updated by the controller of the owner.

Controlled.sol

- changeController: The Controller's new address can be set by the current controller of the owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership of the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of [Etherscan](#) web links. And we have used all possible tests based on given objects as files. We observed 3 low and 6 informational issues in the smart contracts. And those issues are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The Security State of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Theta Token
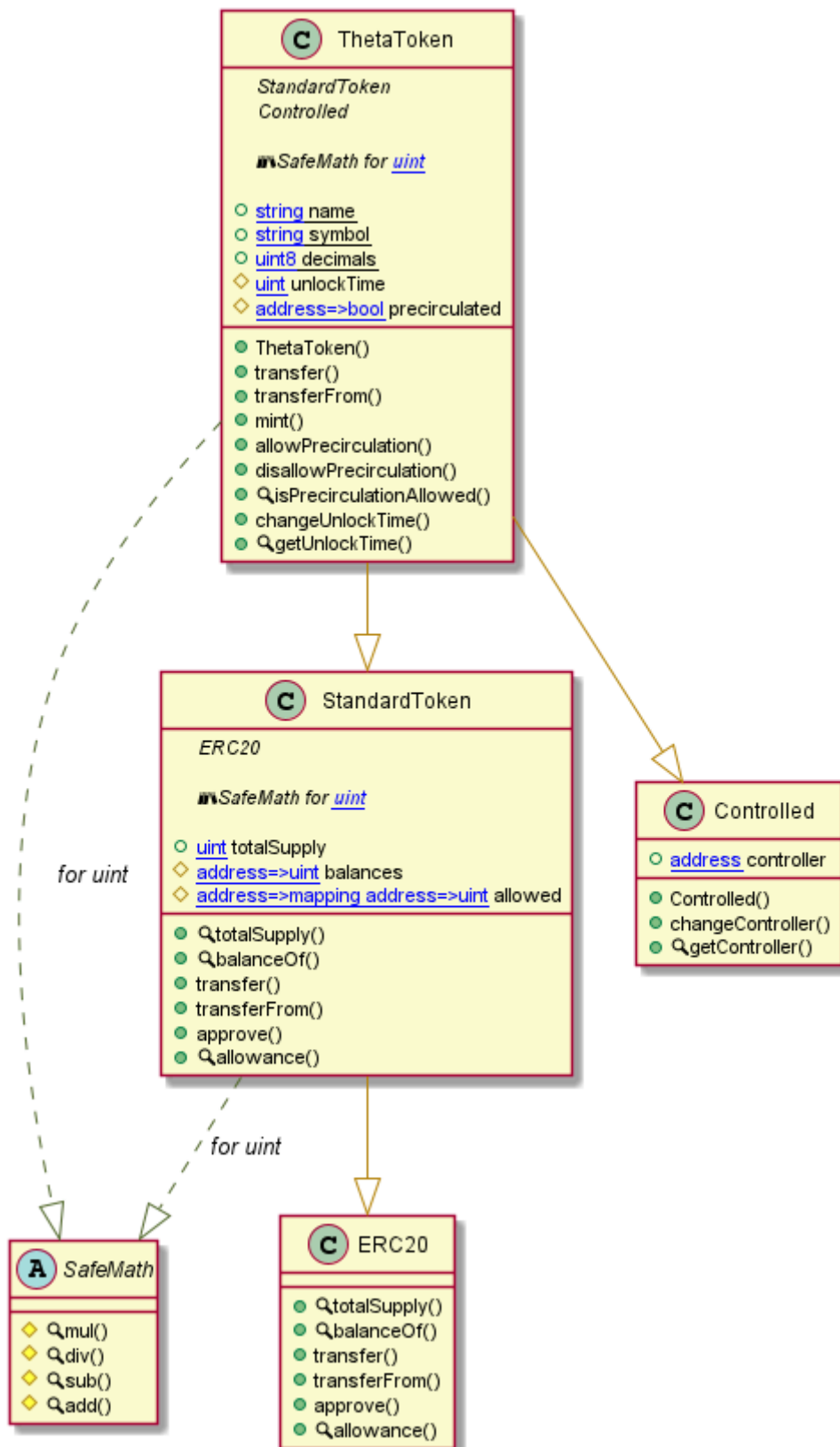
Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> ThetaToken.sol

```
Controlled.changeController(address) (ThetaToken.sol#124-126) should emit an event for:
        - controller = _newController (ThetaToken.sol#125)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

ThetaToken.changeUnlockTime(uint256) (ThetaToken.sol#188-190) should emit an event for:
        - unlockTime = _unlockTime (ThetaToken.sol#189)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Controlled.changeController(address)._newController (ThetaToken.sol#124) lacks a zero-check on :
                - controller = _newController (ThetaToken.sol#125)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

SafeMath.div(uint256,uint256) (ThetaToken.sol#22-27) is never used and should be removed
SafeMath.mul(uint256,uint256) (ThetaToken.sol#13-20) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.4.18 (ThetaToken.sol#5) allows old versions
solc-0.4.18 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Parameter StandardToken.balanceOf(address)._owner (ThetaToken.sol#75) is not in mixedCase
Parameter StandardToken.transfer(address,uint256)._to (ThetaToken.sol#79) is not in mixedCase
Parameter StandardToken.transfer(address,uint256)._value (ThetaToken.sol#79) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._from (ThetaToken.sol#89) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._to (ThetaToken.sol#89) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._value (ThetaToken.sol#89) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._spender (ThetaToken.sol#100) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._value (ThetaToken.sol#100) is not in mixedCase
Parameter StandardToken.allowance(address,address)._owner (ThetaToken.sol#110) is not in mixedCase
Parameter StandardToken.allowance(address,address)._spender (ThetaToken.sol#110) is not in mixedCase
Parameter Controlled.changeController(address)._newController (ThetaToken.sol#124) is not in mixedCase
Modifier Controlled.only_controller() (ThetaToken.sol#132-135) is not in mixedCase
Parameter ThetaToken.transfer(address,uint256)._to (ThetaToken.sol#160) is not in mixedCase
Parameter ThetaToken.transfer(address,uint256)._amount (ThetaToken.sol#160) is not in mixedCase
Parameter ThetaToken.transferFrom(address,address,uint256)._from (ThetaToken.sol#164) is not in mixedCase
Parameter ThetaToken.transferFrom(address,address,uint256)._to (ThetaToken.sol#164) is not in mixedCase
Parameter ThetaToken.transferFrom(address,address,uint256)._amount (ThetaToken.sol#164) is not in mixedCase
Parameter ThetaToken.mint(address,uint256)._owner (ThetaToken.sol#168) is not in mixedCase
Parameter ThetaToken.mint(address,uint256)._amount (ThetaToken.sol#168) is not in mixedCase
Parameter ThetaToken.allowPrecirculation(address)._addr (ThetaToken.sol#176) is not in mixedCase
Parameter ThetaToken.disallowPrecirculation(address)._addr (ThetaToken.sol#180) is not in mixedCase
Parameter ThetaToken.isPrecirculationAllowed(address)._addr (ThetaToken.sol#184) is not in mixedCase
Parameter ThetaToken.changeUnlockTime(uint256)._unlockTime (ThetaToken.sol#188) is not in mixedCase
Modifier ThetaToken.can_transfer(address,address) (ThetaToken.sol#196-199) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
ThetaToken.sol analyzed (5 contracts with 84 detectors), 31 result(s) found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

ThetaToken.sol

Gas costs:

Gas requirement of function ThetaToken.transfer is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 79:4:

Gas costs:

Gas requirement of function ThetaToken.mint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 168:4:

Constant/View/Pure functions:

ThetaToken.transfer(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 160:4:

Similar variable names:

StandardToken.balanceOf(address) : Variables have very similar names "balance" and "balances". Note: Modifiers are currently not considered by this static analysis.
Pos: 76:15:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 133:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 24:13:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

ThetaToken.sol

```
Compiler version ^0.4.18 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:4
Explicitly mark visibility of state
Pos: 5:66
Explicitly mark visibility of state
Pos: 5:68
Provide an error message for require
Pos: 9:79
Provide an error message for require
Pos: 9:89
Provide an error message for revert
Pos: 13:102
Visibility modifier must be first in list of modifiers
Pos: 39:127
Provide an error message for require
Pos: 9:132
Constant name must be in capitalized SNAKE_CASE
Pos: 5:143
Constant name must be in capitalized SNAKE_CASE
Pos: 5:145
Constant name must be in capitalized SNAKE_CASE
Pos: 5:147
Explicitly mark visibility of state
Pos: 5:150
Visibility modifier must be first in list of modifiers
Pos: 80:159
Visibility modifier must be first in list of modifiers
Pos: 68:179
Visibility modifier must be first in list of modifiers
Pos: 62:183
Visibility modifier must be first in list of modifiers
Pos: 65:187
Visibility modifier must be first in list of modifiers
Pos: 39:191
Provide an error message for require
Pos: 9:196
```

Software analysis result:

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.