Cyberscope

Audit Report
Staking

March 2023

SHA256 303b520bf3e0b110acdf9f317f518ae082155990877785e752d23ca232f49c9d
60bc740d665a115c668173ae450d69f9d8b83ea0f62c1506e6d0cc9836333d64

Audited by © cyberscope

Table of Contents

Review

Testing Deploy

Filename	Explorer
Referral.sol	https://testnet.bscscan.com/address/0xde94a125b79858862c264f5b1fbc2eee08937a0b
staking.sol	https://testnet.bscscan.com/address/0xe06A9FdE09397A8366f35dCBFd4A32ad1F3526Bb

Audit Updates

Initial Audit	13 Feb 2023
Corrected Phase 2	15 Mar 2023

Source Files

Filename	SHA256
Referral.sol	60bc740d665a115c668173ae450d69f9d8b83ea0f62c1506e6d0cc9836333d64
staking.sol	303b520bf3e0b110acdf9f317f518ae082155990877785e752d23ca232f49c9d

Introduction

This audit is focused on the Staking and the Referral contract.

Staking

The Staking contract implements a staking mechanism. Users can stake tokens in order to obtain rewards.

Roles

The contract consists of an owner role.

The `Owner` has the authority to:

- Set fee address.
- Update emision rate.
- Set referral commission rate.
- Add liquidity pool.
- Configure allocation point and deposit fee of a pool.

The `Users` have the authority to:

- View pending reward Aeternas.
- Mass update pools.
- Update a specific pool.
- Deposit tokes to a liquidity pool.
- Withdraw tokens from a liquidity pool.
- Emergency withdraw tokens from a liquidity pool.

Referral

The Referral contract implements a referral mechanism for the staking contract.

Roles

The contract consists of an owner and an operator role.

The `Owner` roles have the authority to grant or revoke the operator role.

The `Operator` role has the authority to record a referral.

The `Users` have the authority to view the recorded referrals.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	MSC	Missing Sanity Check	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	staking.sol
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The function arguments are not properly sanitized.

```
function set(uint256 pid, uint256 allocPoint, uint16 depositFeeBP)
external onlyOwner {...}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

- The `_depositFeeBP` should be lower than 10000.
- The variable multiplication `pool.allocPoint` should be lower than the totalAllocPoint.

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	staking.sol#L311
Status	Unresolved

Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
uint256 public totalAllocPoint = 0;
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	staking.sol#L366,367
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
PoolInfo storage poo
UserInfo storage use
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions, because it is unnecessary and can make the code less readable.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	staking.sol#L327
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
aetern
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	staking.sol#L338,352,495,505
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
totalAllocPoint = totalAllocPoint.add(allocPoint)
totalAllocPoint =
totalAllocPoint.sub(poolInfo[pid].allocPoint).add(allocPoint)
aeternaPerBlock = newAeternaPerBlock
referralCommissionRate = newReferralCommissionRate
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	staking.sol#L137,145,153,157,224,231,236 Referral.sol#L186,213,239,249,264,274,279,390,394,405,416,421,432
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function sendValue(address payable recipient, uint256 amount) internal
{
        require(address(this).balance >= amount, "Address: insufficient
balance");

        // solhint-disable-next-line avoid-low-level-calls,
avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may
have reverted");
...
function functionCall(address target, bytes memory data) internal
returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call
failed");
    }

function functionCallWithValue(address target, bytes memory data,
uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address:
low-level call with value failed");
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	staking.sol#L371,372,397,398
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 aeternaReward =
(multiplier.mul(aeternaPerBlock).mul(pool.allocPoint)).div(totalAllocPo
int)
accAeternaPerShare =
accAeternaPerShare.add((aeternaReward.mul(1e18)).div(pool.lpSupply))
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	staking.sol#L133,175 Referral.sol#L193,292
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { codehash := extcodehash(account) }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-

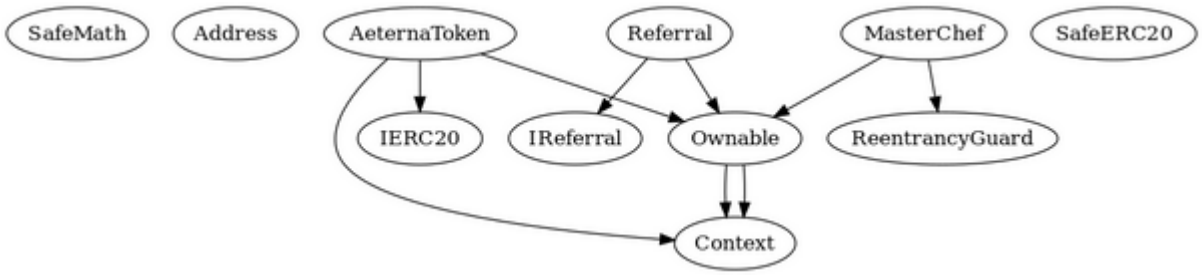
	approve	External	✓	-
	transferFrom	External	✓	-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Internal	✓	
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IReferral	Interface			
	recordReferral	External	✓	-
	getReferrer	External		-
Referral	Implementation	IReferral, Ownable		
	recordReferral	External	✓	onlyOperator
	getReferrer	Public		-
	updateOperator	External	✓	onlyOwner

Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Internal	✓	
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	

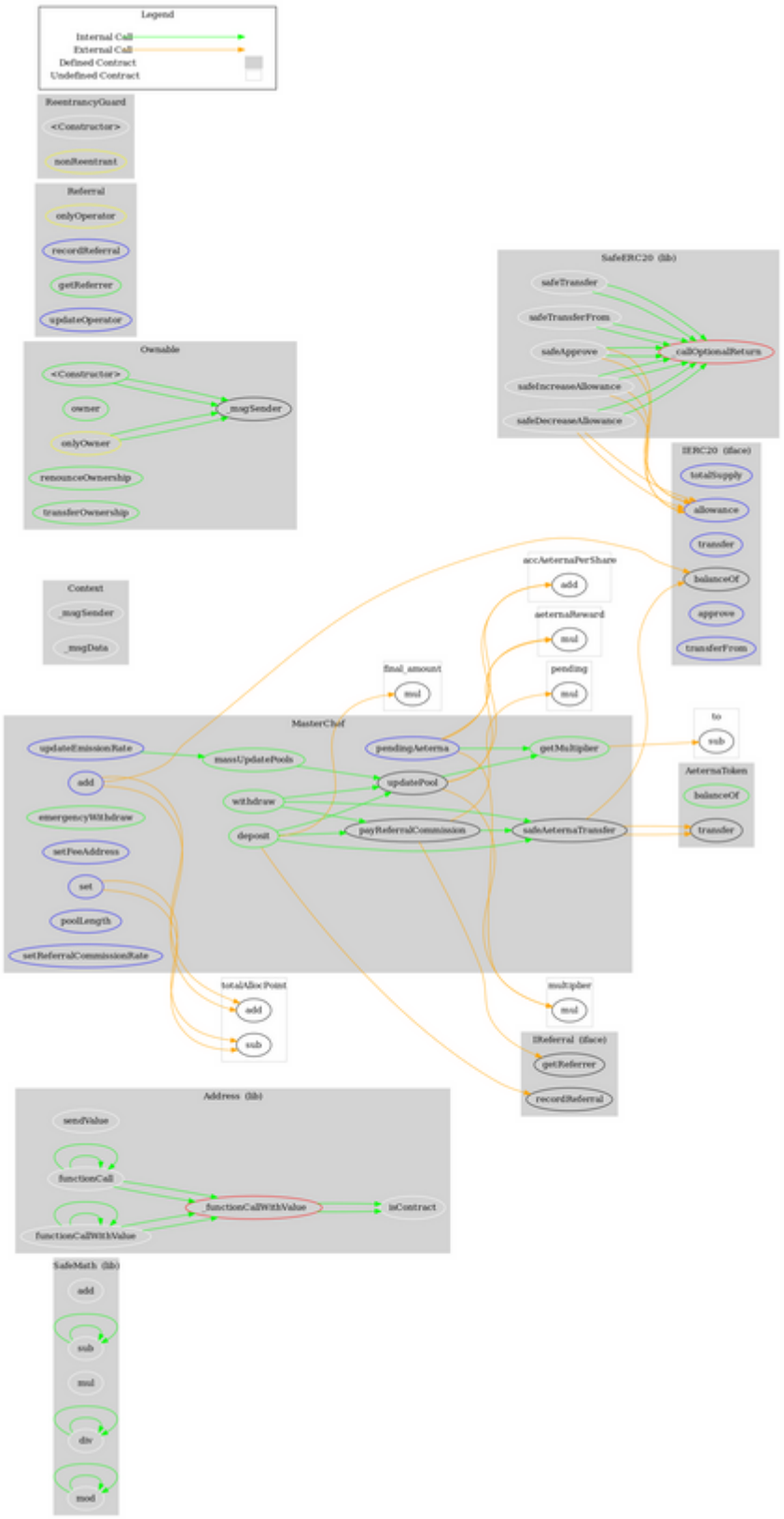
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
AeternaToken	Implementation	Context, IERC20, Ownable		
	balanceOf	Public		-
	transfer	Public	✓	-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
ReentrancyGuard	Implementation			
		Internal	✓	
IReferral	Interface			
	recordReferral	External	✓	-
	getReferrer	External		-

MasterChef	Implementation	Ownable, Reentrancy Guard		
		Public	✓	-
	add	External	✓	onlyOwner
	set	External	✓	onlyOwner
	getMultiplier	Public		-
	pendingAeterna	External		-
	massUpdatePools	Public	✓	-
	updatePool	Public	✓	-
	deposit	Public	✓	nonReentrant
	withdraw	Public	✓	nonReentrant
	emergencyWithdraw	Public	✓	nonReentrant
	safeAeternaTransfer	Internal	✓	
	setFeeAddress	External	✓	onlyOwner
	updateEmissionRate	External	✓	onlyOwner
	poolLength	External		-
	setReferralCommissionRate	External	✓	onlyOwner
	payReferralCommission	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ReentrancyGuard
<Constructor>
nonReentrant
Referral
onlyOperator
recordReferral
getReferrer
updateOperator
Ownable
<Constructor>
owner
onlyOwner
renounceOwnership
transferOwnership
_msgSender
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
safeIncreaseAllowance
safeDecreaseAllowance
callOptionalReturn
IERC20 (iface)
totalSupply
allowance
transfer
balanceOf
approve
transferFrom
Context
_msgSender
_msgData
accAeternaPerShare
add
aeternaReward
mul
pending
mul
final_amount
mul
MasterChef
updateEmissionRate
add
emergencyWithdraw
setFeeAddress
set
poolLength
setReferralCommissionRate
massUpdatePools
withdraw
deposit
pendingAeterna
updatePool
payReferralCommission
getMultiplier
safeAeternaTransfer
to
sub
AeternaToken
balanceOf
transfer
totalAllocPoint
add
sub
multiplier
mul
IReferral (iface)
getReferrer
recordReferral
Address (lib)
sendValue
functionCall
functionCallWithValue
functionCallWithValue
isContract
SafeMath (lib)
add
sub
mul
div
mod

Summary

Staking contract implements a token and staking mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io