Cyberscope

Audit Report

Staking

February 2023

SHA256 b288f566431bc294170211ea7e3ff3147e0066cb258694c12aa36073d6fced1c
53dc92813376e279ec9e23b5cac11053676bc73b3e22647dc8051acf61351e38

Audited by © cyberscope

Table of Contents

Review

Testing Deploy

Filename	Explorer
Referral.sol	https://testnet.bscscan.com/address/0xb3D99db3C5ADD294ffde44443FDe5c5Bd0f24281
staking.sol	https://testnet.bscscan.com/address/0xfDc487c941aE5D6Cd2868aEe0215F4C917141b8F

Audit Updates

Initial Audit	13 Feb 2023

Source Files

Filename	SHA256
Referral.sol	53dc92813376e279ec9e23b5cac11053676bc73b3e22647dc8051acf61351e38
staking.sol	b288f566431bc294170211ea7e3ff3147e0066cb258694c12aa36073d6fced1c

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
●	MCAC	Missing Constructor Argument Check	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

MCAC - Missing Constructor Argument Check

Criticality	Minor / Informative
Location	staking.sol#L310
Status	Unresolved

Description

The contract initializes variables that have not been properly checked on the constructor. These variables may produce vulnerability issues.

```
constructor(
    AeternaToken _Aeterna,
    address _feeAddress,
    IReferral _referral
) public {
    Aeterna = _Aeterna;
    feeAddress = _feeAddress;
    referral=_referral;
}
```

Recommendation

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

- The addresses should not be set to zero address.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	staking.sol#L416,481
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 `AeternaPerBlock` variable is not properly sanitized.

```
function updateEmissionRate(uint256 _AeternaPerBlock) external
onlyOwner {
    massUpdatePools();
    AeternaPerBlock = _AeternaPerBlock;
}
```

The argument `_depositFeeBP` should be set to values higher than the denominator of the percentage.

```
function add(IERC20 _lpToken, uint256 _allocPoint, uint16
_depositFeeBP) external onlyOwner {
    _lpToken.balanceOf(address(this));
    uint256 lastRewardBlock = block.number;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(PoolInfo({
        lpToken: _lpToken,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        accAeternaPerShare: 0,
        depositFeeBP: _depositFeeBP,
        lpSupply: 0
    }));
}
```

Recommendation

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

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

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	staking.sol#L301,305,330,345,354,359,380,397,430,449,465,476,481,491,497 Referral.sol#L557,570,575
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
AeternaToken public Aeterna
uint256 public AeternaPerBlock
IERC20 _lpToken
uint16 _depositFeeBP
uint256 _allocPoint
uint256 _pid
uint256 _from
uint256 _to
address _user
address _referrer
uint256 _amount
address _to
address _feeAddress
uint256 _AeternaPerBlock

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	staking.sol#L333,346,483,493
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 = _AeternaPerBlock
referralCommissionRate = _referralCommissionRate
```

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#L365,366,391,392
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(totalAllocPoin
t)
pool.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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	staking.sol#L325
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
feeAddress = _feeAddress
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

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		-

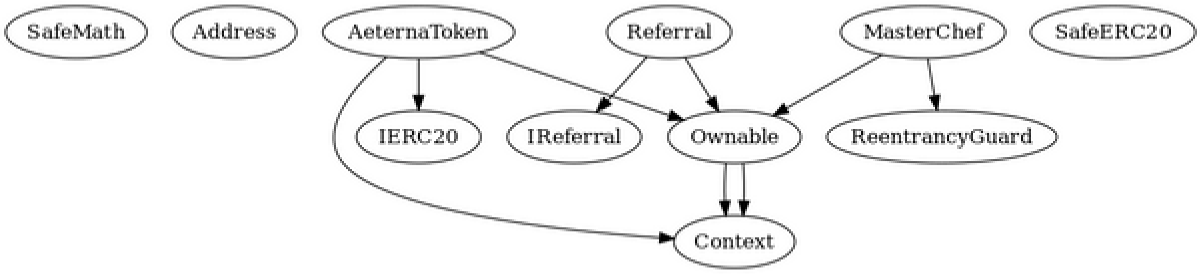
	approve	External	✓	-
	transferFrom	External	✓	-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
Context	Implementation			
	_msgSender	Internal		
	_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	✓	

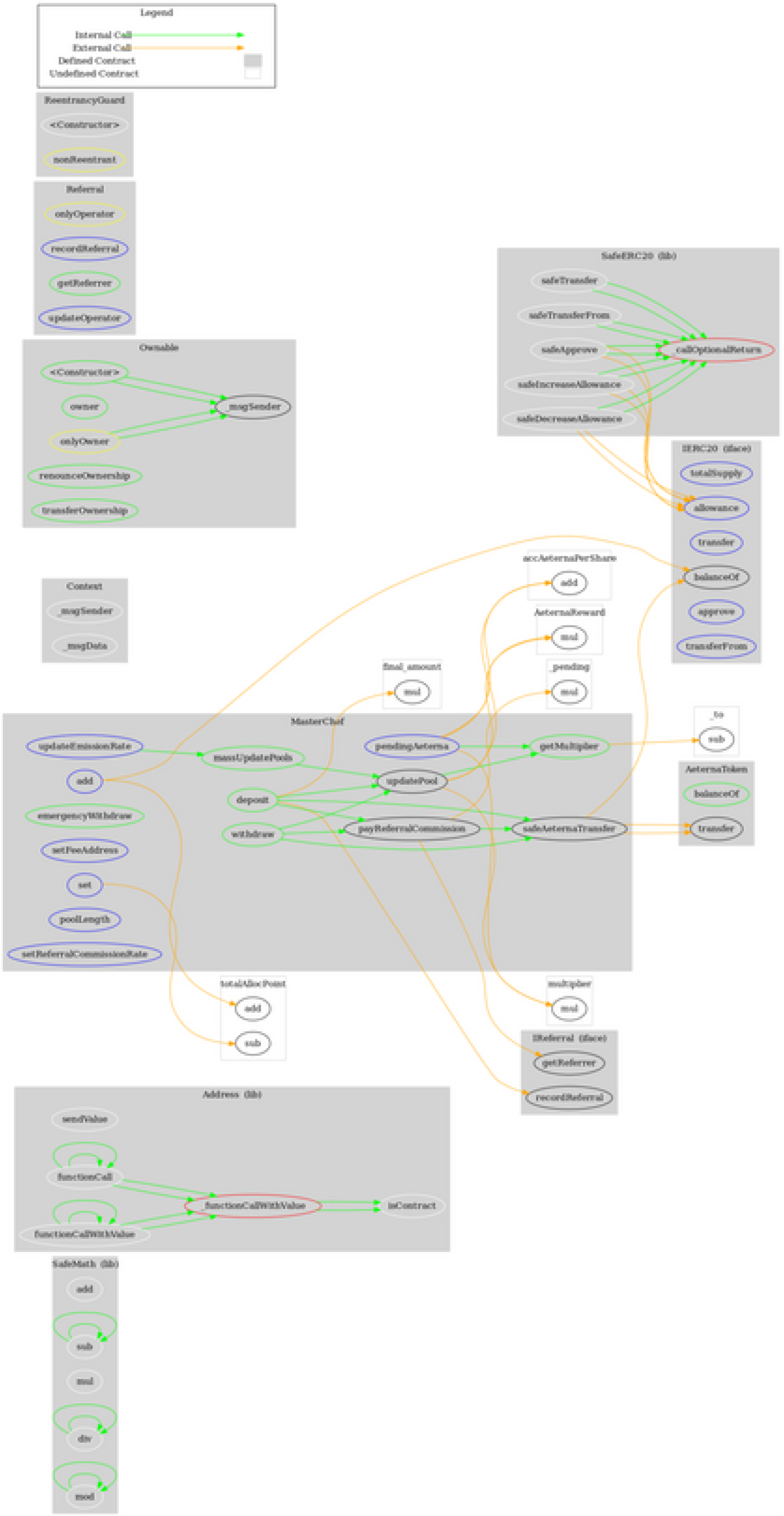
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
AeternaToken	Implementation	Context, IERC20, Ownable		
	balanceOf	Public		-
	transfer	Public	✓	-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	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
Context
_msgSender
_msgData
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
safeIncreaseAllowance
safeDecreaseAllowance
callOptionalReturn
IERC20 (iface)
totalSupply
allowance
transfer
balanceOf
approve
transferFrom
accAeternaPerShare
add
AeternaReward
mul
final_amount
mul
_pending
mul
_to
sub
MasterChef
updateEmissionRate
add
emergencyWithdraw
setFeeAddress
set
poolLength
setReferralCommissionRate
massUpdatePools
deposit
withdraw
pendingAeterna
updatePool
payReferralCommission
getMultiplier
safeAeternaTransfer
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