Cyberscope

Audit Report

ArchieNeko

February 2023

Commit	5ecf04e5609a01486681ba2d1c645a29bd6f4bfb
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/archieneko/ARCHIE-CHAIN-staking-contracts
Commit	5ecf04e5609a01486681ba2d1c645a29bd6f4bfb

Audit Updates

Initial Audit	20 Feb 2023

Source Files

Filename	SHA256
MockStaker.sol	7ac03347dfa2a4f0603b1b3686dbd3b0b3d12ef1ba0d9b6a1c6504f4c25c8aa3
Staking.sol	3765212850f013c37db0266e91e4a52acb70cadacc66d57aa2ecdac57ead15e8

Introduction

The Staking contract allows users to stake their Ethereum and become validators. The contract has a set of minimum and maximum validator thresholds and allows users become a validator in the network when their staked amount reaches at least 100,000 ether, with validators being responsible for validating transactions and adding them to the blockchain. The contract also allows validators to register their BLS public key and provides functions to stake, unstake, and check staking information. The contract is designed to prevent multiple staking and to limit the number of validators in the network, with the number of validators being between the minimum and maximum thresholds set during contract deployment.

Roles

EOA

The EOA role can interact with the following functions:

- `receive()`
- `function stake()`
- `function unstake()`

Staker

The Staker role can interact with the following functions:

- `function unstake()`

Validator

The Validator role does not interact with any function.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	FUA	Function Unrestricted Access	Unresolved
●	MC	Missing Check	Unresolved
●	UM	Unused Modifier	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L19	Stable Compiler Version	Unresolved

FUA - Function Unrestricted Access

Criticality	Minor / Informative
Location	Staking.sol#L106
Status	Unresolved

Description

The contract's `registerBLSPublicKey()` function adds a public key to the `_addressToBLSPublicKey` variable. The `validatorBLSPublicKeys()` function returns only the registered keys of the `validators`. The `registerBLSPublicKey()` function can be called by any user, not just validators. As a result, the `validatorBLSPublicKeys()` function may return inaccurate results.

```
function validatorBLSPublicKeys() public view returns (bytes[] memory) {
    bytes[] memory keys = new bytes[](_validators.length);

    for (uint256 i = 0; i < _validators.length; i++) {
        keys[i] = _addressToBLSPublicKey[_validators[i]];
    }

    return keys;
}
...
function registerBLSPublicKey(bytes memory blsPubKey) public {
    _addressToBLSPublicKey[msg.sender] = blsPubKey;

    emit BLSPublicKeyRegistered(msg.sender, blsPubKey);
}
```

Recommendation

The team is advised to restrict the access of the `registerBLSPublicKey()` function only to the `validators`.

MC - Missing Check

Criticality	Minor / Informative
Location	MockStaker.sol#L10
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 `_staking` variable is a reference to the address of the `Staking` contract, which is provided as an argument during the instantiation of the contract. It is important to note that the contract does not perform any validation on the input value of `_staking`, which means that it is possible for an incorrect or malicious value to be passed in during instantiation.

```
_staking = Staking(stakingContractAddr);
```

Recommendation

The team is advised to properly check the variables according to the required specifications, in order to avoid any unexpected behavior or security vulnerabilities in the contract's execution.

UM - Unused Modifier

Criticality	Minor / Informative
Location	Staking.sol#L44
Status	Unresolved

Description

A modifier is a special type of function in Solidity that can be used to modify the behavior of other functions. When a modifier is applied to a function, it can perform certain checks or modifications to the input parameters or state variables of the function before it is executed.

An unused modifier means that there is a defined modifier in the contract code that is not actually being used in any of the functions. This can be an indication of a mistake or oversight in the development of the contract. It is generally good practice to remove any unused code in a smart contract to improve its readability and reduce the potential for bugs or security vulnerabilities.

```
modifier onlyValidator() {
    require(_isValidator(msg.sender), "Only validator can call function");
    _;
}
```

Recommendation

To avoid creating unused modifiers, it's important to carefully consider the modifiers that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Staking.sol#L12,14,15,16,17,18,19,21 MockStaker.sol#L7
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.

```
address[] public _validators
mapping(address => bool) public _addressToIsValidator
mapping(address => uint256) public _addressToStakedAmount
mapping(address => uint256) public _addressToValidatorIndex
uint256 public _stakedAmount
uint256 public _minimumNumValidators
uint256 public _maximumNumValidators
mapping(address => bytes) public _addressToBLSPublicKey
Staking public _staking
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Staking.sol#L1 MockStaker.sol#L1
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.7;
```

Recommendation

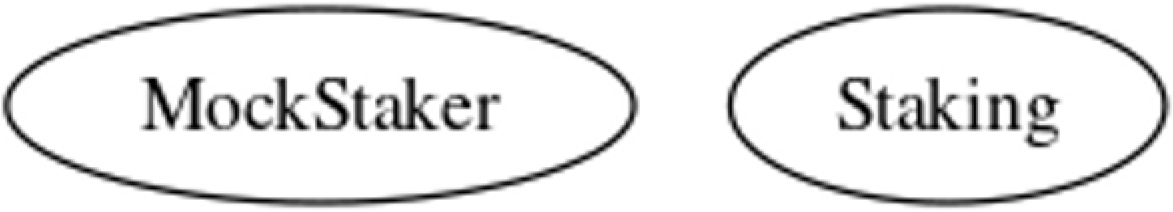
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

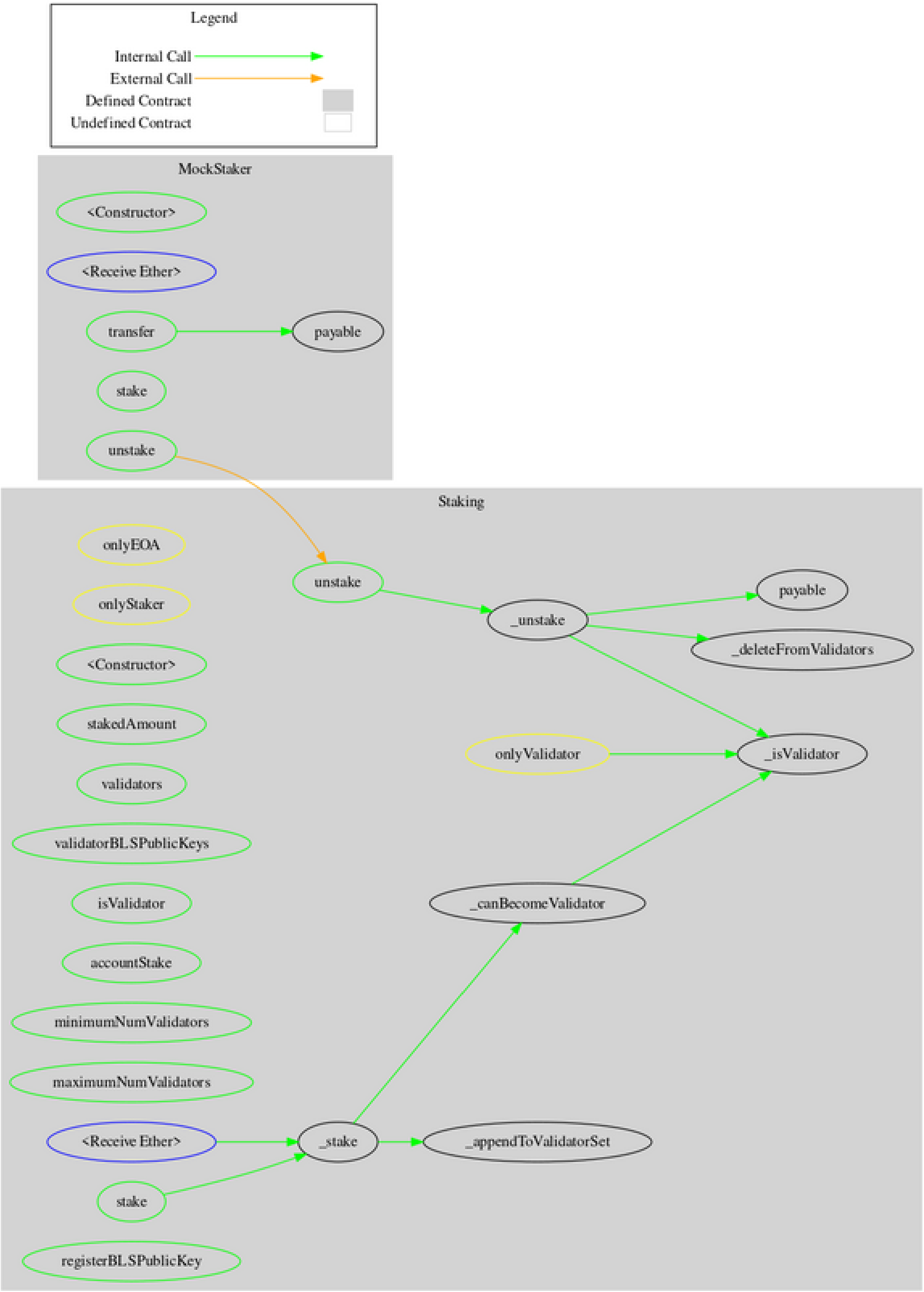
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MockStaker	Implementation			
		Public	✓	-
		External	Payable	-
	transfer	Public	✓	-
	stake	Public	✓	-
	unstake	Public	✓	-
Staking	Implementation			
		Public	✓	-
	stakedAmount	Public		-
	validators	Public		-
	validatorBLSPublicKeys	Public		-
	isValidator	Public		-
	accountStake	Public		-
	minimumNumValidators	Public		-
	maximumNumValidators	Public		-
		External	Payable	onlyEOA
	stake	Public	Payable	onlyEOA
	unstake	Public	✓	onlyEOA onlyStaker
	registerBLSPublicKey	Public	✓	-
	_stake	Private	✓	
	_unstake	Private	✓	
	_deleteFromValidators	Private	✓	

	_appendToValidatorSet	Private	✓	
	_isValidator	Private		
	_canBecomeValidator	Private		

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
MockStaker
<Constructor>
<Receive Ether>
transfer
payable
stake
unstake
Staking
onlyEOA
onlyStaker
<Constructor>
stakedAmount
validators
validatorBLSPublicKeys
isValidator
accountStake
minimumNumValidators
maximumNumValidators
<Receive Ether>
stake
registerBLSPublicKey
unstake
_unstake
payable
_deleteFromValidators
onlyValidator
_isValidator
_canBecomeValidator
_stake
_appendToValidatorSet

Summary

Staking contract implements a 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