Cyberscope

Audit Report

OpSec

April 2024

SHA 256 700f7da31ad5ef8ce79ef283747d9f52df1656636683ff37e681714cd2bff59c

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/opseccloud/staking
Commit	d3046fedc4d24a46b93cc7a3e6b59cdae76fb530
Testing Deploy	https://testnet.bscscan.com/address/0xbc501934ece4c8732710acd5857a16e402b8a181

Audit Updates

Initial Audit	26 Apr 2024

Source Files

Filename	SHA256
contracts/OpsecStaking.sol	700f7da31ad5ef8ce79ef283747d9f52df1656636683ff37e681714cd2bff59c
@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol	de9a20a313d34dd3b454c834606518390c8ead4dcca8137c879401fef62d8583
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	a08e16324da33a9d666dc07a22ae58031c242a3869f6808e55b4b82fc70cb209
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	a8b7eafa0fdc7cb5a644c8c61a8e4c51e031d5e1e6f268f72dbe18b768ead56e
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	9247b9ad7939d23990dbdc9274917c3762ffb37e5137ef7bbfcc2e2fba1b8dd2
@openzeppelin/contracts/utils/Address.sol	b3710b1712637eb8c0df81912da3450da6ff67b0b3ed18146b033ed15b1aa3b9
@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee

@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	471157c89111d7b9eab456b53ebe9042bc69504a64cb5cc980d38da9103379ae
@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol	912509e0e9bf74e0f8a8c92d031b5b26d2d35c6d4abf3f56251be1ea9ca946bf

Overview

The OpsecStaking contract is designed to facilitate the staking of ERC20 tokens, specifically the token referred to within the contract as `opsec`. It leverages the OpenZeppelin libraries, including safe token handling and upgradeable contract features.

The primary functionality of the OpsecStaking contract allows users to stake their tokens to earn rewards over a specified duration. The staking process is managed through unique identifiers for each staking instance, termed `stakeId`. This identifier ensures that each staking record is unique and traceable. Users initiate a stake by specifying the amount of tokens and the duration for which these tokens will be locked. The contract records the timestamp when the stake is created, which is used to manage the lock period and validate unstaking requests.

Additionally, the contract provides functionality for users to extend the duration of their existing stakes. This feature allows users to continue their investment beyond the initial term, potentially to receive greater rewards. Unstaking is similarly user-driven, where users can retrieve their staked tokens after the lock period has expired.

Administrative functions are also included, which are restricted to the contract's owner. These functions allow the contract to be paused or unpaused. The owner also has the capability to distribute Ethereum stored in the contract and withdraw tokens.

The contract integrates events for each critical action — staking, unstaking, extending the stake duration, and claims — which aid in tracking transactions and changes in contract state on the blockchain.

Upgradeability

The OpsecStaking contract is deployed as an upgradeable proxy. This choice allows for the implementation contract, to be upgraded. While this architecture provides significant adaptability benefits, it also poses certain risks, as the functionalities and the logic of the contract can change substantially.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CCR	Contract Centralization Risk	Unresolved
	MU	Modifiers Usage	Unresolved
	PTAI	Potential Transfer Amount Inconsistency	Unresolved
	SLM	Staking Logic Misalignment	Unresolved
	USTR	User Staked Tokens Risk	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L19	Stable Compiler Version	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	contracts/OpsecStaking.sol#L66,155,177
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion. Specifically, the `claim` function requires the contract owner to initiate transactions, deciding which users receive Ethereum payouts and how much each receives. This setup places substantial control in the hands of a single entity, potentially allowing for arbitrary or biased distributions of rewards. Additionally, The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the withdraw function. Lastly, the owner can stop the functioning of the contract by calling the `pause` function.

```
function pause() external onlyOwner {
    _pause();
}

function claim(
    uint256[] calldata amounts,
    address[] calldata users
) external onlyOwner whenNotPaused {
    require(
        amounts.length == users.length,
        "Invalid the length of amounts and users"
    );

    for (uint256 i = 0; i < amounts.length; i++) {
        payable(users[i]).sendValue(amounts[i]);

        emit Claimed(amounts[i], users[i]);
    }
}

function withdraw(
    address token,
    address user,
    uint256 amount
) external onlyOwner {
    IERC20(token).safeTransfer(user, amount);
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	contracts/OpsecStaking.sol#L120,137
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(stakeData.user == msg.sender, "You are not the
staker");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	contracts/OpsecStaking.sol#L96,147
Status	Unresolved

Description

The `safeTransfer()` and `safeTransferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
opsec.safeTransferFrom(msg.sender, address(this), amount);

opsec.safeTransfer(msg.sender, stakeData.amount);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
 Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

SLM - Staking Logic Misalignment

Criticality	Minor / Informative
Location	contracts/OpsecStaking.sol#L83,135,155
Status	Unresolved

Description

The contract primarily functions as a token locker rather than a traditional staking contract, due to its reward distribution mechanism. In typical staking contracts, rewards are often calculated and distributed automatically based on the amount and duration of the tokens staked by each user. However, in this contract, the claim function, which is responsible for distributing rewards, must be manually invoked by the contract owner. This function allows the owner to arbitrarily decide both the recipients and the amounts of the distributions. There is no inherent mechanism within the contract that automatically calculates and allocates staking rewards based on predefined criteria such as staking duration or amounts. This setup significantly deviates from the usual expectations of a staking mechanism where rewards are expected to be distributed in a decentralized and algorithmically defined manner.

```
function claim(
    uint256[] calldata amounts,
    address[] calldata users
) external onlyOwner whenNotPaused {
    require(
        amounts.length == users.length,
        "Invalid the length of amounts and users"
    );

    for (uint256 i = 0; i < amounts.length; i++) {
        payable(users[i]).sendValue(amounts[i]);

        emit Claimed(amounts[i], users[i]);
    }
}
```

Recommendation

It is recommended to reconsider the logic around the staking mechanisms, in order align the contract more closely with standard staking model expectations, such as automatic reward calculation and distribution mechanism. This mechanism should compute rewards based on the amount of tokens staked and the duration of the stake, directly linking the staking activity with its rewards. Implementing such changes would not only reduce the need for manual intervention by the contract owner but also minimize the risk of biased reward distribution, thereby promoting a fairer and more transparent staking process.

USTR - User Staked Tokens Risk

Criticality	Minor / Informative
Location	contracts/OpsecStaking.sol#L135,177
Status	Unresolved

Description

As described in the CCR finding, the contract owner has the authority to withdraw any tokens from the contract without restrictions. This functionality can lead to scenarios where users are unable to unstake their tokens because the necessary funds have been removed from the contract.

```
function unstake(bytes32 stakeId) external whenNotPaused {
    StakeData storage stakeData = stakes[stakeId];
    require(stakeData.user == msg.sender, "You are not the
staker");
    require(!stakeData.unstaked, "You already unstaked");
    require(
        stakeData.timestamp + stakeData.duration <=
block.timestamp,
        "The stake is still locked"
    );

    stakeData.unstaked = true;
    stakeAmounts[msg.sender] -= stakeData.amount;

    opsec.safeTransfer(msg.sender, stakeData.amount);

    emit Unstaked(stakeId, msg.sender, stakeData.amount,
block.timestamp);
}
```

Recommendation

The team should follow the recommendations of the CCR finding and adding to that user staked tokens should be safeguarded from potential misuse.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/OpsecStaking.sol#L56
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.

```
IERC20 _opsec
```

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	contracts/OpsecStaking.sol#L2
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 ^0.8.24;
```

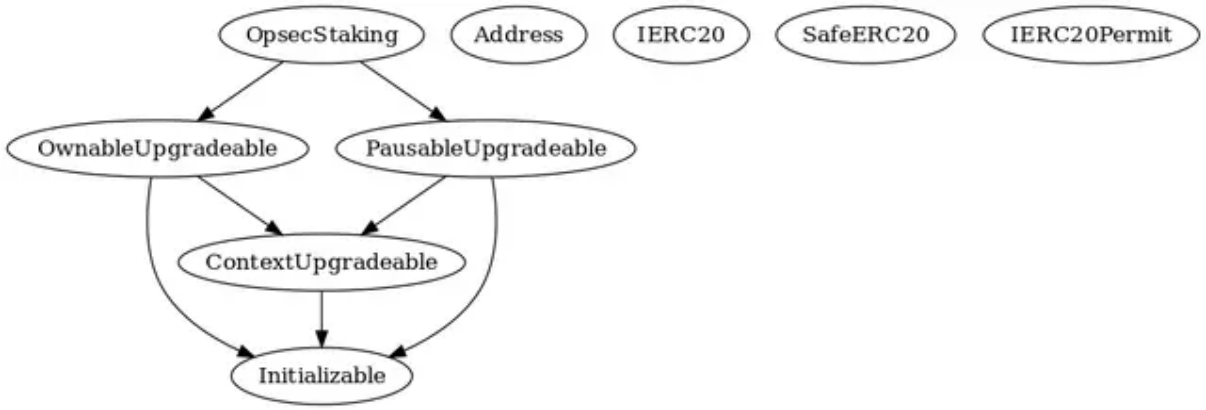
Recommendation

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.

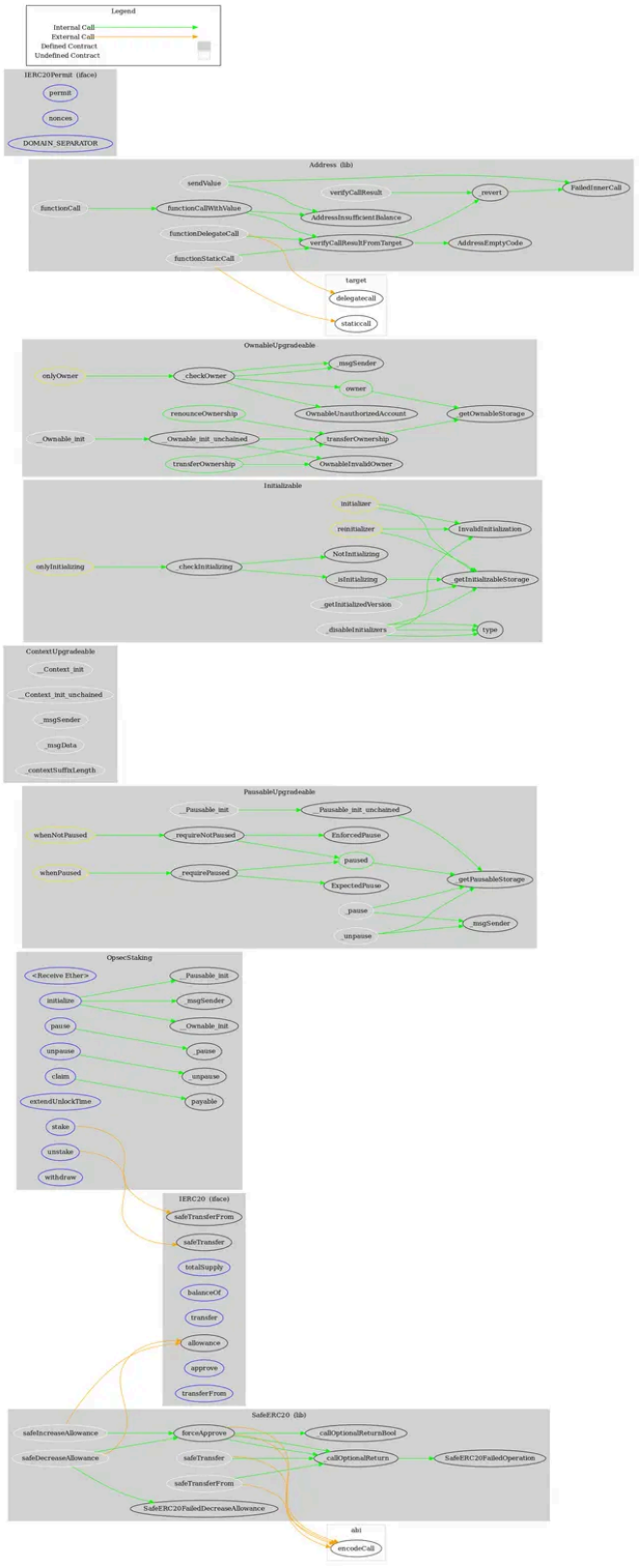
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
OpsecStaking	Implementation	OwnableUpgradeable, PausableUpgradeable		
		External	Payable	-
	initialize	External	✓	initializer
	pause	External	✓	onlyOwner
	unpause	External	✓	onlyOwner
	stake	External	✓	whenNotPaused
	extendUnlockTime	External	✓	whenNotPaused
	unstake	External	✓	whenNotPaused
	claim	External	✓	onlyOwner whenNotPaused
	withdraw	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR
Address (lib)
sendValue
functionCall
functionCallWithValue
functionDelegateCall
functionStaticCall
verifyCallResult
AddressInsufficientBalance
verifyCallResultFromTarget
revert
FailedInnerCall
AddressEmptyCode
target
delegatecall
staticcall
OwnableUpgradeable
onlyOwner
checkOwner
_msgSender
owner
renounceOwnership
_Ownable_init
_Ownable_init_unchained
transferOwnership
OwnableUnauthorizedAccount
OwnableInvalidOwner
getOwnableStorage
Initializable
initializer
reinitializer
onlyInitializing
checkInitializing
NotInitializing
isInitializing
_getInitializedVersion
_disableInitializers
InvalidInitialization
getInitializableStorage
type
ContextUpgradeable
__Context_init
__Context_init_unchained
_msgSender
_msgData
_contextSuffixLength
PausableUpgradeable
_Pausable_init
_Pausable_init_unchained
whenNotPaused
requireNotPaused
whenPaused
requirePaused
EnforcedPause
paused
ExpectedPause
_pause
_unpause
_msgSender
getPausableStorage
OpsecStaking
<Receive Ether>
initialize
pause
unpause
claim
extendUnlockTime
stake
unstake
withdraw
_Pausable_init
_msgSender
_Ownable_init
_pause
_unpause
payable
IERC20 (iface)
safeTransferFrom
safeTransfer
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
SafeERC20 (lib)
safeIncreaseAllowance
safeDecreaseAllowance
forceApprove
safeTransfer
safeTransferFrom
SafeERC20FailedDecreaseAllowance
callOptionalReturnBool
callOptionalReturn
SafeERC20FailedOperation
abi
encodeCall

Summary

OpSec 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