Cyberscope

Audit Report

Rosy token

March 2024

Files BurntSteakDeployer.sol, Orchestrator.sol, Steak.sol, Burnt.sol, Carbon.sol

Audited by © cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x083e269e3cbf7d1279254b435cd311466f0b04ad

Audit Updates

Initial Audit	06 Mar 2024 https://github.com/cyberscope-io/audits/blob/main/rosy/v1/audit.pdf
Corrected Phase 2	14 Mar 2024

Source Files

Filename	SHA256
contracts/Steak.sol	e7875ddf832afb3516244134de820c7fda1f63289a5328b6a595ff02be02d1cf
contracts/Orchestrator.sol	34884c030ec8f4f38d9d226e2a2453a8256dac27b399fc15925c714b8738c280
contracts/IBurnableERC20.sol	a3609f701cb0e462b1aaf2e0c13994d88c371175c1cdee2d9c87b2697aaade8b
contracts/Carbon.sol	ad266fa20c19de1cdb9555be032a52567dcd88991a03d5daf9bd65192184145a
contracts/BurntSteakDeployer.sol	3026e2e18675a635a35a0f47e79946f4f00146b4577603b6362d1d8bd869c1be
contracts/Burnt.sol	4fd1853accc77ae69529e1684def62918cd93342e0519d5a92d75efcdab24543
@openzeppelin/contracts/utils/Context.sol	847fda5460fee70f56f4200f59b82ae622bb03c79c77e67af010e31b7e2cc5b6

@openzeppelin/contracts/utils/math/Math.sol	a6ee779fc42e6bf01b5e6a963065706e882b016affbedfd8be19a71ea48e6e15
@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee
@openzeppelin/contracts/interfaces/IERC20.sol	cb42f0b4d269ba8ef2629c176a7f99bf4fb50837c92f45596b54822b26e3df4b
@openzeppelin/contracts/access/Ownable2Step.sol	90f1f1cdd07ce4b90e987065e82899fdaa6ef967d1996915143c6e39818e160c
@openzeppelin/contracts/access/Ownable.sol	38578bd71c0a909840e67202db527cc6b4e6b437e0f39f0c909da32c1e30cb81
@oasisprotocol/sapphire-contracts/contracts/Sapphire.sol	7b04d3f2de70838e615786cb7fd49e08cbe117c3f42b3c81e024b950385bf484

Overview

BurntSteakDeployer.sol

The `BurntSteakDeployer` contract serves as the entry point for initializing the staking, burning, and rewards ecosystem of the project. It is responsible for deploying and setting up key components of the system, including the `Orchestrator`, `RandomMultiRewardEmitter`, and indirectly, the associated `Burnt`, `Steak`, and `Carbon` contracts through the `Orchestrator`. This contract sets the parameters for the ecosystem, such as the token to be used (rosyToken), burn thresholds, burn rates, and the rewards point rate. Upon deployment, it transfers ownership of the `Orchestrator` to the deployer.

Orchestrator.sol

The `Orchestrator` contract acts as the centre for managing the staking, burning, and rewards components of the project. It inheriting from `Ownable2Step`, adding an extra layer of security for ownership transfers. This contract directly initializes and integrates the `Burnt`, `Steak`, and `Carbon` contracts, setting key parameters for each component based on the initial configuration passed during its own construction.

Access Control Implements custom modifiers like onlySteak and publicBurnAllowed to enforce access control, ensuring that only authorized interactions occur.

Administration and Configuration Provides functions for the contract owner to adjust key operational parameters such as burn thresholds, burn rates, rewards rates, and even the ability to enable or disable public token burning. It also allows for the management of component contract ownership.

Steak.sol

The `Steak` contract is dedicated to the staking functionality within the project, enabling users to stake and unstake tokens as part of their participation in the ecosystem. It is designed to work closely with the Orchestrator contract, signaling stake changes and interacting with other components of the system, particularly for the purpose of adjusting rewards and managing token burns. When stake changes are made, they are accompanied by the emission of `Staked` and `Unstaked` events for transparency and tracking.

Burnt.sol

The `Burnt` contract is designed to manage the burning of tokens within the ecosystem. It introduces a mechanism to burn tokens based on a calculated rate that can adjust over time, influenced by various factors within the system. This contract allows for a responsive approach to token burning. Parameters such as the `burnThreshold`, `baseBurnPerSecond`, `maxBurnPerSecond`, and `scaleFactor` can be adjusted by the contract owner. Emits events to provide transparency over the contract's actions.

Carbon.sol

The `Carbon` contract is integral to the rewards system of the project, focusing on the accumulation and redemption of points based on users' staking and unstakin. It provides a flexible framework for calculating user points over time and converting these points into rewards, facilitating an engaging user experience. Emits events like `PointsRedeemed` to offer transparency and traceability

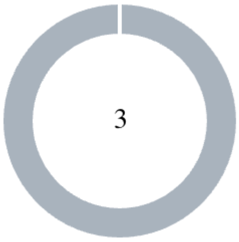
Point Accumulation Implements a mechanism for users to accumulate points over time, based on factors such as the duration of their stake. This is achieved through a combination of the user's points factor and the system's annual rate, allowing for dynamic rewards calculation.

Reward Redemption Offers users the ability to redeem their accumulated points for rewards. The actual redemption process is handled by `rewardEmitter`, which is responsible for determining the rewards given in exchange for points.

Audit Scope

The current audit report is specifically focues on the following contract files: `BurntSteakDeployer.sol` , `Orchestrator.sol` , `Steak.sol` , `Carbon.sol` , `Burnt.sol` . The `RandomMultiRewardEmitter.sol` is out of audit scope for the current audit phase. This means that while the provided contracts are thoroughly examined for security and functionality, any interactions, dependencies, or integrations with the aforementioned contract are not covered in this audit report. This limitation should be taken into consideration when interpepreting the findings and conclusion of this audit.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	3

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	0	3	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Minor / Informative	CCR	Contract Centralization Risk	Acknowledged
Minor / Informative	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
Minor / Informative	RSW	Redundant Storage Writes	Acknowledged

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	contracts/Orchestrator.sol#L74,78,82,86,90,94...
Status	Acknowledged

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 contract owner has the authority to set key variables, that impact the functionality of the contract. This capability grants the contract owner substantial control.

```
function setBurnInfluencingFactor(IBurnInfluencingFactor
_burnInfluencingFactor) external onlyOwner {
    burnt.setBurnInfluencingFactor(_burnInfluencingFactor);
}

function setburnThreshold(uint256 _burnThreshold) external
onlyOwner {
    burnt.setburnThreshold(_burnThreshold);
}

function setAllowPublicBurn(bool _allowPublicBurn) public
onlyOwner {
    allowPublicBurn = _allowPublicBurn;
    emit AllowPublicBurnUpdated(msg.sender, allowPublicBurn);
}
```

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.

Team Update

The team has acknowledged that this is not a security issue and states: *This centralization risk is acceptable as we'd prefer to keep the flexibility to adjust formulas for future promotions or other needs.*

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	contracts/Steak.sol#L36,50
Status	Acknowledged

Description

The `transfer()` and `transferFrom()` 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

```
stakingToken.transferFrom(msg.sender, address(this), amount)
stakingToken.transfer(msg.sender, 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
```

Team Update

The team has acknowledged that this is not a security issue and states: *The staking token, in our case ROSY, doesn't have tax so transfer amounts will be consistent.*

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/Burnt.sol#L95,100,105,111,117 contracts/Carbon.sol#L107,112,117 contracts/Steak.sol#L61 contracts/Orchestrator.sol#L114
Status	Acknowledged

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function setBurnInfluencingFactor(IBurnInfluencingFactor
_burnInfluencingFactor) external onlyOwner {
    burnInfluencingFactor = _burnInfluencingFactor;
    emit BurnInfluencingFactorUpdated(msg.sender,
address(burnInfluencingFactor));
}

function setburnThreshold(uint256 _burnThreshold) external
onlyOwner {
    burnThreshold = _burnThreshold;
    emit BurnThresholdUpdated(msg.sender, burnThreshold);
}

function setMaxBurnPerSecond(uint256 _maxBurnPerSecond)
external onlyOwner {
    require(_maxBurnPerSecond >= baseBurnPerSecond, "Max burn
rate can't be less than base rate");
    maxBurnPerSecond = _maxBurnPerSecond;
    emit MaxBurnPerSecondUpdated(msg.sender, maxBurnPerSecond);
}
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

Team Update

The team has acknowledged that this is not a security issue and states: *Since the owner will be a human responsible for any of those changes, they are responsible for ensuring they don't waste gas with a redundant storage write.*

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IStakeChangeListener	Interface			
	onBeforeStakeChange	External	✓	-
Steak	Implementation	Ownable		
		Public	✓	Ownable
	stake	External	✓	-
	unstake	External	✓	-
	_onBeforeStakeChange	Internal	✓	
	setStakeChangeListener	External	✓	onlyOwner
Orchestrator	Implementation	Ownable2Step, IBurnInfluencingFactor, IUserPointsFactor, IStakeChangeListener		
		Public	✓	Ownable
	getBurnInfluencingFactor	External		-
	getUserPointsFactor	External		-
	onBeforeStakeChange	External	✓	onlySteak
	tryBurn	External	Payable	publicBurnAllowed

	setBurnInfluencingFactor	External	✓	onlyOwner
	setburnThreshold	External	✓	onlyOwner
	setBaseBurnPerSecond	External	✓	onlyOwner
	setMaxBurnPerSecond	External	✓	onlyOwner
	setScaleFactor	External	✓	onlyOwner
	withdraw	External	✓	onlyOwner
	setStakeChangeListener	External	✓	onlyOwner
	setUserPointsFactor	External	✓	onlyOwner
	setRewardEmitter	External	✓	onlyOwner
	setAnnualRateBasisPoints	External	✓	onlyOwner
	setAllowPublicBurn	Public	✓	onlyOwner
	setPublicBurnFee	Public	✓	onlyOwner
	withdrawPublicBurnFee	External	✓	onlyOwner
	transferComponentOwnership	Public	✓	onlyOwner
	renounceComponentOwnership	Public	✓	onlyOwner
IUserPointsFactor	Interface			
	getUserPointsFactor	External		-
IRewardEmitter	Interface			
	onBeforeUpdatePoints	External	✓	-
	redeemPoints	External	✓	-

Carbon	Implementation	Ownable		
		Public	✓	Ownable
	_getAnnualRatePerSecond	Internal		
	_getUserPointsFactor	Internal		
	getEarnedPointsSinceLastUpdate	Public		-
	currentPoints	External		-
	updatePoints	External	✓	onlyOwner
	_updatePoints	Internal	✓	
	redeemPoints	External	✓	-
	setUserPointsFactor	External	✓	onlyOwner
	setRewardEmitter	External	✓	onlyOwner
	setAnnualRateBasisPoints	External	✓	onlyOwner
BurntSteakDeployer	Implementation			
		Public	✓	-
IBurnInfluencingFactor	Interface			
	getBurnInfluencingFactor	External		-
Burnt	Implementation	Ownable		
		Public	✓	Ownable
	_getBurnInfluencingFactor	Internal		
	burnRatePerSecond	Public		-

	tryBurn	External	✓	onlyOwner
	setBurnInfluencingFactor	External	✓	onlyOwner
	setburnThreshold	External	✓	onlyOwner
	setBaseBurnPerSecond	External	✓	onlyOwner
	setMaxBurnPerSecond	External	✓	onlyOwner
	setScaleFactor	External	✓	onlyOwner
	withdraw	External	✓	onlyOwner

Summary

Rosy token implements a staking, token burning and rewards 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