Cyberscope

Audit Report
Quidax Cards

February 2025

SHA

6f711fbf08ff75cbe7ec7c9a3307471a36832125aaa5e217e3e249a95f216f2d

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://gitlab.com/QuidaxAdmin/raffle-smart-contract
Commit	81383ca040b66897d72bd73f903141af71a11df9

Audit Updates

Initial Audit	18 Feb 2025

Source Files

Filename	SHA256
Cards.sol	6f711fbf08ff75cbe7ec7c9a3307471a36832125aaa5e217e3e249a95f216f2d

Overview

Cards Contract

The `Cards` contract is an ERC1155 multi-token contract with supply tracking and access control. It is designed for a collectible upgradeable card system where users can mint, upgrade, and burn cards. The contract supports role-based access management through OpenZeppelin's `AccessControl`, allowing administrators and minters to control token issuance and upgrades. It also includes an upgrade system where tokens can be burned to obtain higher-tier cards. The contract is initialized with a default administrator who is granted the `DEFAULT_ADMIN_ROLE`. The administrator can set upgrade costs for different card levels. The contract supports a predefined set of card tiers:

- `SAFARI`
- `STELLAR`
- `GOLD`
- `DIAMOND`

Token Minting

The `issueNew` function allows accounts with the `MINTER` role to mint new `SAFARI` tokens, which are the base-level cards in the system. Only accounts with the `MINTER` role can invoke this function.

Token Upgrades

The `upgradeToken` function enables users to upgrade their cards to a higher tier. This is done by burning a specified number of lower-tier tokens and receiving a new upgraded token. The contract ensures that the user has a sufficient balance before processing the upgrade. The `_upgradeToken` internal function handles the burning and minting logic while emitting an `Upgraded` event upon success.

Setting Upgrade Costs

The contract allows the administrator to define upgrade costs through `setUpgradeCosts`. The `_setUpgradeCosts` internal function enforces that exactly

three upgrade cost configurations are provided. Users can check the cost of an upgrade using the `upgradeCost` function.

Token Metadata

The contract allows administrators to update token URIs using the `setURI` function. Each token can have a unique `URI` stored in a private mapping `_uris`, which is returned when calling the `uri` function.

Burning Tokens

The `burnWinningTokens` function allows minters to burn a specified amount of `SAFARI` tokens. This function is designed for integration with external raffle contract that require token burning.

Total Supply Tracking

The contract implements supply tracking using `ERC1155Supply`. The totals function returns the total supply of all four card tiers.

Access Control

The contract implements OpenZeppelin's `AccessControl` to manage roles, ensuring that only authorized entities can execute certain functions.

Roles

Administrator

The `DEFAULT_ADMIN_ROLE` is the highest authority in the contract, responsible for setting parameters such as upgrade costs and token metadata.

- `setUpgradeCosts(Upgrade[] memory upgradeCosts)`
- `setURI(uint256 tokenId, string memory newUri)`

Minter

The `MINTER` role is responsible for issuing new tokens and burning tokens when required.

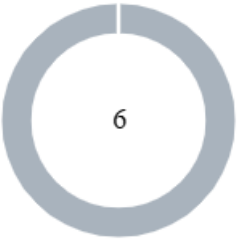
- `issueNew(address account, uint256 amount)`
- `burnWinningTokens(address account, uint256 amount)`

Token Holders

Any address holding `QDX-Cards` can interact with standard ERC1155 functions, including transfers and upgrades.

- `upgradeToken(uint256 toToken, uint256 amount)`

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	MEE	Missing Events Emission	Unresolved
	BT	Burns Tokens	Unresolved
	CC	Commented Code	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	MC	Missing Check	Unresolved
	L19	Stable Compiler Version	Unresolved

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Cards.sol#L89,203
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setURI(
    uint256 tokenId,
    string memory newUri
) public onlyRole(DEFAULT_ADMIN_ROLE) {
    _uris[tokenId] = newUri;
}

function _setUpgradeCosts(Upgrade[] memory upgradeCosts)
internal {
    require(upgradeCosts.length == 3, "Invalid upgrade costs
length");
    for (uint256 i = 2; i < 5; i++) {
        _upgradeCost[i] = upgradeCosts[i - 2];
    }
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

BT - Burns Tokens

Criticality	Minor / Informative
Location	Cards.sol#L211
Status	Unresolved

Description

A `MINTER` has the authority to burn tokens from a specific address. They may take advantage of it by calling the `burnWinningTokens` function. As a result, the targeted address will lose the corresponding tokens. If the role is granted to an address that is not part of the intended governance plan, it could lead to severe abuse.

```
function burnWinningTokens(
    address account,
    uint256 amount
) external onlyRole(MINTER) {
    _burn(account, SAFARI, amount);
}
```

Recommendation

The team should carefully manage the assignment of the MINTER role to prevent unauthorized access and potential abuse. Only trusted addresses should be granted this privilege, and any role assignment should undergo thorough review and approval, preferably through a decentralized governance mechanism.

Additionally the team may consider:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Otherwise, the team may use the common pattern where users have to first approve certain entities to burn their tokens.

CC - Commented Code

Criticality	Minor / Informative
Location	Cards.sol#L39,50,179
Status	Unresolved

Description

The contract has instances where functions or variables have been commented out. This may hinder code readability.

```
// function setTopCadre(...){...}
// bytes32 public constant ADMIN = keccak256("ADMIN");
// function setPremiumRewards(...){...}
```

Recommendation

It is recommended to remove unnecessary comments to maximize code readability.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	Cards.sol#L77,89,172,211
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.

```
function setUpgradeCosts(Upgrade[] memory upgradeCosts)
external onlyRole(DEFAULT_ADMIN_ROLE) {}

function setURI(uint256 tokenId, string memory newUri) public
onlyRole(DEFAULT_ADMIN_ROLE) {}

function issueNew(address account, uint256 amount) public
override onlyRole(MINTER) {}

function burnWinningTokens(address account, uint256 amount)
external onlyRole(MINTER) {}
```

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.

MC - Missing Check

Criticality	Minor / Informative
Location	Cards.sol#L203
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.

Specifically in `_setUpgradeCosts` that is called in the `constructor` during deployment and in `setUpgradeCosts` , used by the default admin role, a check is missing to ensure that each `Upgrade.costToken` in the `upgradeCosts` array is within the range of `SAFARI=1` and `GOLD=3` .

```
function _setUpgradeCosts(Upgrade[] memory upgradeCosts)
internal {
    require(upgradeCosts.length == 3, "Invalid upgrade costs
length");
    for (uint256 i = 2; i < 5; i++) {
        _upgradeCost[i] = upgradeCosts[i - 2];
    }
}
```

Recommendation

The team is advised to properly check the `Upgrade.costToken` is within valid ranges according to the required specifications.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Cards.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 ^0.8.28;
```

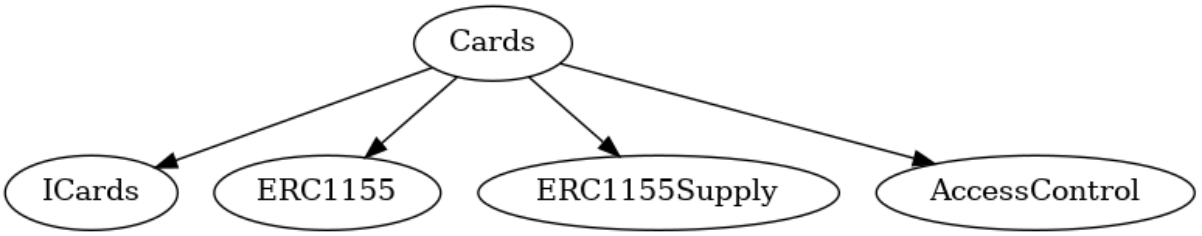
Recommendation

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**
Cards	Implementation	ICards, ERC1155, ERC1155Supply, AccessControl		
		Public	✓	ERC1155
	setUpgradeCosts	External	✓	onlyRole
	setURI	Public	✓	onlyRole
	uri	Public		-
	name	External		-
	symbol	External		-
	upgradeToken	External	✓	-
	_upgradeToken	Internal	✓	
	issueNew	Public	✓	onlyRole
	upgradeCost	Public		-
	_setUpgradeCosts	Internal	✓	
	burnWinningTokens	External	✓	onlyRole
	totals	External		-
	supportsInterface	Public		-
	_update	Internal	✓	
	totalSupply	Public		-

Inheritance Graph



Summary

Quidax Cards contract implements an ERC115 token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Quidax Cards is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues.

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

cyberscope.io