# RugFreeCoins Audit

# Epic Hero Reflection

# Smart Contract Security Audit

# September 15, 2021

# Contents

# Audit details

**Audited project**

Epic Hero Reflection Token

**Contract Address**

0x09eAf2a4bcE29796EE380Aae6a3D23B817Ad67EB

**Client contact**

Epic Hero Token Team

**Blockchain**

Binance smart chain

**Project website**

https://epichero.io/

# Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Rugfreecoins and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Rugfreecoins) owe no duty of care towards you or any other person, nor does Rugfreecoins make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Rugfreecoins hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Rugfreecoins hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Rugfreecoins, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

# Background

Rugfreecoins was commissioned by Epic Hero to perform an audit of the smart contract.

**https://bscscan.com/address/0x09eAf2a4bcE29796EE380Aae6a3D23B817Ad67EB**

The focus of this audit is to verify that the smart contract is secure, resilient and working according to the specifications.

The information in this report should be used to understand the risk exposure of the smart contract, project feasibility, long term sustainability and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

# About the project

The fee of 1.25% when buying and 3.75% when selling from the main Epic Hero will be deducted and sent to the Epic Hero reflection contract where automatic WBNB rewards will be distributed among NFT holders proportional to how many tokens each individual holds. This innovative mechanism gives the EpicHero NFT holders great passive income over time simply by holding the NFT in their wallets.

So those not even engaging in epic gaming need not worry about their NFT value because simply by holding their NFT cards they can gain passive income over time in the form of BNB, and this feature will make EpicHero NFT card become more desirable and more popular between non-gaming users.

To facilitate EpicHero NFT passive reflection rewards, a 10% tax is levied on each EpicHero NFT transaction made in the Thoreum NFT Marketplace. Half of all these taxes are returned to the remaining NFT holders, in the form of BNB reflections, the other half, funds game development, advertising, and marketing expenses.

**Benefits**

- NFT reflection rewards in BNB: It means the longer the holders hold NFTs the bigger the amount of dividend they get in BNB, it gives holders the incentive to hold to collect the dividend and with highly limited numbers, this feature makes the NFT look more attractive in the long run because everyone desires to have it.

- Not only is Epic Hero NFT highly limited in numbers, but also it will become rarer and rarer day by day because of our NFT merging (burning) mechanism. Users can choose 2 heroes to merge into a new 1 with better attributes or rarity, so the number of heroes will be decreased a lot over time.

# Contract details

## Token contract details for 15th September 2021

| | |
|---|---|
| **Contract address** | 0x09eAf2a4bcE29796EE380Aae6a3D23B817Ad67EB |
| **EpicHeroNFt address** | 0xafdcb0ecad1c8cb22893dca7d6c510dbfda3bbec |
| **WBNB address** | 0xbb4cdb9cbd36b01bd1cbaebf2de08d9173bc095c |
| **Contract deployer address** | 0x8E377Cc27aBfB273313791097bcCe590a84F1F97 |
| **Contract's current owner address** | 0x8e377cc27abfb273313791097bcce590a84f1f97 |

# Contract code function details

| No | Category | Item | Result |
|---|---|---|---|
| 1 | Coding conventions | BRC20 Token standards | pass |
| | | compile errors | pass |
| | | Compiler version security | pass |
| | | visibility specifiers | pass |
| | | Gas consumption | pass |
| | | SafeMath features | pass |
| | | Fallback usage | pass |
| | | tx.origin usage | pass |
| | | deprecated items | pass |
| | | Redundant code | pass |
| | | Overriding variables | pass |
| 2 | Function call audit | Authorization of function call | pass |
| | | Low level function (call/delegate call) security | pass |
| | | Returned value security | pass |
| | | Selfdestruct function security | pass |
| 3 | Business security | Access control of owners | pass |
| | | Business logics | pass |
| | | Business implementations | pass |
| 4 | Integer overflow/underflow | | pass |
| 5 | Reentrancy | | pass |
| 6 | Exceptional reachable state | | pass |
| 7 | Transaction ordering dependence | | pass |
| 8 | Block properties dependence | | pass |
| 9 | Pseudo random number generator (PRNG) | | pass |
| 10 | DoS (Denial of Service) | | pass |
| 11 | Token vesting implementation | | pass |
| 12 | Fake deposit | | pass |
| 13 | Event security | | pass |

# Contract description table

Below table represents the summary of the contracts and methods in the token contract. We scanned the whole contract and listed down all the Interfaces, functions and implementations with its visibility and mutability.

| Contract | Type | Bases | | |
|---|---|---|---|---|
| └ | Function Name | Visibility | Mutability | Modifiers |
| | | | | |
| **IBEP20** | **Interface** | | | |
| └ | totalSupply | External ❗ | | NO❗ |
| └ | decimals | External ❗ | | NO❗ |
| └ | symbol | External ❗ | | NO❗ |
| └ | name | External ❗ | | NO❗ |
| └ | getOwner | External ❗ | | NO❗ |
| └ | balanceOf | External ❗ | | NO❗ |
| └ | transfer | External ❗ | 🛑 | NO❗ |
| └ | allowance | External ❗ | | NO❗ |
| └ | approve | External ❗ | 🛑 | NO❗ |
| └ | transferFrom | External ❗ | 🛑 | NO❗ |
| | | | | |
| **IERC721Receiver** | **Interface** | | | |

| | | | | |
|---|---|---|---|---|
| └ | onERC721Received | External ❗ | ◉ | NO❗ |
| | | | | |
| **IERC721** | **Interface** | | | |
| └ | ownerOf | External ❗ | | NO❗ |
| └ | safeTransferFrom | External ❗ | ◉ | NO❗ |
| └ | balanceOf | External ❗ | | NO❗ |
| | | | | |
| **IERC721Enumerable** | **Interface** | **IERC721** | | |
| └ | totalSupply | External ❗ | | NO❗ |
| └ | tokenOfOwnerByIndex | External ❗ | | NO❗ |
| └ | tokenByIndex | External ❗ | | NO❗ |
| | | | | |
| **IEpicHeroNFT** | **Interface** | **IERC721Enumerable** | | |
| └ | packIdOfToken | External ❗ | | NO❗ |
| └ | getHero | External ❗ | | NO❗ |
| | | | | |
| **EpicAuth** | **Implementation** | | | |
| └ | | Public ❗ | ◉ | NO❗ |
| └ | authorizeFor | Public ❗ | ◉ | authorizedFor |
| └ | authorizeForMultiplePermissions | Public ❗ | ◉ | authorizedFor |

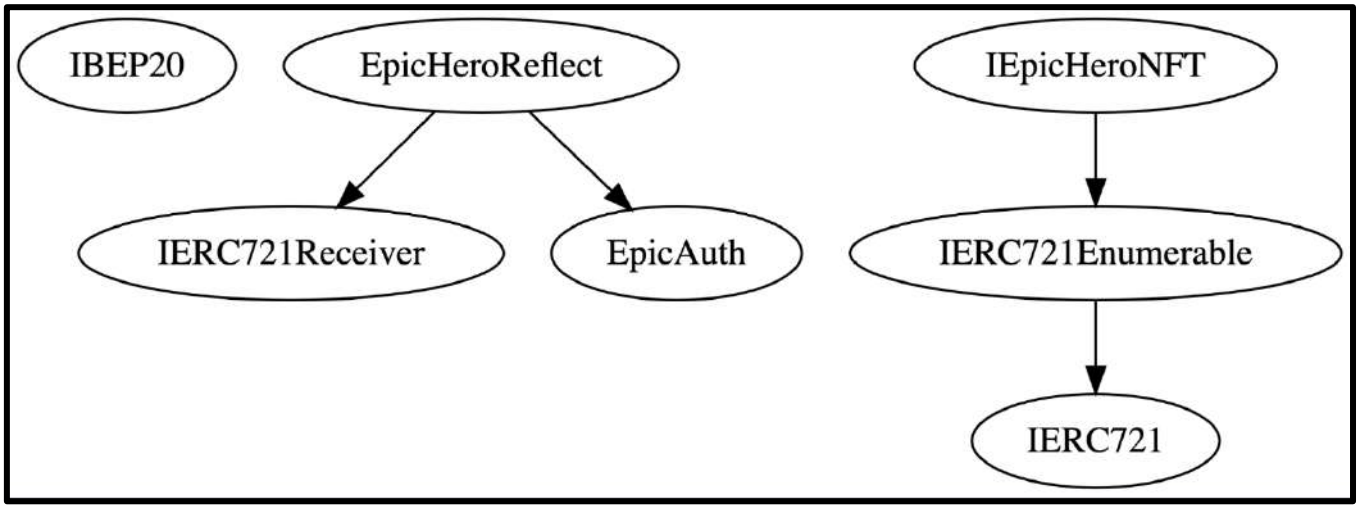
| | | | | |
|---|---|---|---|---|
| L | unauthorizeFor | Public ❗ | ◉ | authorizedFor |
| L | unauthorizeForMultiplePermissions | Public ❗ | ◉ | authorizedFor |
| L | isOwner | Public ❗ | | NO❗ |
| L | isAuthorizedFor | Public ❗ | | NO❗ |
| L | isAuthorizedFor | Public ❗ | | NO❗ |
| L | transferOwnership | Public ❗ | ◉ | onlyOwner |
| L | getPermissionNameToIndex | Public ❗ | | NO❗ |
| L | getPermissionUnlockTime | Public ❗ | | NO❗ |
| L | isLocked | Public ❗ | | NO❗ |
| L | lockPermission | Public ❗ | ◉ | authorizedFor |
| L | unlockPermission | Public ❗ | ◉ | NO❗ |
| | | | | |
| **EpicHeroReflect** | **Implementation** | **EpicAuth, IERC721Receiver** | | |
| L | | Public ❗ | ◉ | EpicAuth |
| L | initializeMints | Public ❗ | ◉ | authorizedFor |
| L | currentRate | Public ❗ | | NO❗ |
| L | claimAllRewards | External ❗ | ◉ | NO❗ |
| L | claimAllRewards | Public ❗ | ◉ | NO❗ |

| | | | | |
|---|---|---|---|---|
| L | claimReward | Public ! | ● | NO! |
| L | getAllUnrealizedRewards | Public ! | | NO! |
| L | getAllUnrealizedRewards | Public ! | | NO! |
| L | getUnrealizedReward | Public ! | | NO! |
| L | getRealizedRewards | Public ! | | NO! |
| L | getRealizedReward | Public ! | | NO! |
| L | reflectDividend | Internal 🔒 | ● | |
| L | updateRewards | Public ! | ● | NO! |
| L | registerNewMint | External ! | ● | authorized For |
| L | batchRegisterMints | External ! | ● | authorized For |
| L | updateBurnedToken | External ! | ● | authorized For |
| L | setWbnbAddress | External ! | ● | authorized For |
| L | setEpicHeroNFTAddress | External ! | ● | authorized For |
| L | teamSetDividend | External ! | ● | authorized For |
| L | retrieveTokens | External ! | ● | authorized For |
| L | retrieveBNB | External ! | ● | authorized For |
| L | onERC721Received | Public ! | | NO! |
| L | _setDividend | Internal 🔒 | ● | |

***Legend***

| Symbol | Meaning |
|---|---|
| | **Function can modify state** |
| | **Function is payable** |

## Inheritance Hierarchy




# Security issue checking status

- **High severity issues**
  - **No high severity issues found.**
- **Medium severity issues**
  - **No medium severity issues found.**
- **Low severity issues**
  - **No low severity issues found.**

# Owner privileges

- The owner can authorize permissions for wallets.

```
/**
 * Authorize address for one permission
 */
ftrace | funcSig
function authorizeFor(address adr, string memory permissionName)
    public
    authorizedFor(Permission.Authorize)
{
    uint256 permIndex = permissionNameToIndex[permissionName];
    authorizations[adr][permIndex] = true;
    emit AuthorizedFor(adr, permissionName, permIndex);
}

/**
 * Authorize address for multiple permissions
 */
ftrace | funcSig
function authorizeForMultiplePermissions(
    address adr,
    string[] calldata permissionNames
) public authorizedFor(Permission.Authorize) {
    for (uint256 i; i < permissionNames.length; i++) {
        uint256 permIndex = permissionNameToIndex[permissionNames[i]];
        authorizations[adr][permIndex] = true;
        emit AuthorizedFor(adr, permissionNames[i], permIndex);
    }
}
```

❖ The owner can unauthorize permission for wallets.

```
/**
 * Remove address' authorization
 */
ftrace | funcSig
function unauthorizeFor(address adr, string memory permissionName)
    public
    authorizedFor(Permission.Unauthorize)
{
    require(adr != owner, "Can't unauthorize owner");

    uint256 permIndex = permissionNameToIndex[permissionName];
    authorizations[adr][permIndex] = false;
    emit UnauthorizedFor(adr, permissionName, permIndex);
}

/**
 * Unauthorize address for multiple permissions
 */
ftrace | funcSig
function unauthorizeForMultiplePermissions(
    address adr,
    string[] calldata permissionNames
) public authorizedFor(Permission.Unauthorize) {
    require(adr != owner, "!owner");

    for (uint256 i; i < permissionNames.length; i++) {
        uint256 permIndex = permissionNameToIndex[permissionNames[i]];
        authorizations[adr][permIndex] = false;
        emit UnauthorizedFor(adr, permissionNames[i], permIndex);
    }
}
```

❖ The owner can transfer ownership.

```
/**
 * Transfer ownership to new address. Caller must be owner.
 */
ftrace | funcSig
function transferOwnership(address payable adr) public onlyOwner {
    address oldOwner = owner;
    owner = adr;
    for (uint256 i; i < NUM_PERMISSIONS; i++) {
        authorizations[oldOwner][i] = false;
        authorizations[owner][i] = true;
    }
    emit OwnershipTransferred(oldOwner, owner);
}
```

❖ The owner can lock and unlock permissions.

```
/*
 *Locks the permission from being used for the amount of time provided
 */
ftrace | funcSig
function lockPermission(string memory permissionName, uint64 time)
    public
    virtual
    authorizedFor(Permission.LockPermissions)
{
    uint256 permIndex = permissionNameToIndex[permissionName];
    uint64 expiryTime = uint64(block.timestamp) + time;
    lockedPermissions[permIndex] = PermissionLock(true, expiryTime);
    emit PermissionLocked(permissionName, permIndex, expiryTime);
}

/*
 * Unlocks the permission if the lock has expired
 */
ftrace | funcSig
function unlockPermission(string memory permissionName) public virtual {
    require(
        block.timestamp > getPermissionUnlockTime(permissionName),
        "TimeLock"
    );
    uint256 permIndex = permissionNameToIndex[permissionName];
    lockedPermissions[permIndex].isLocked = false;
    emit PermissionUnlocked(permissionName, permIndex);
}
```

```
ftrace | funcSig
function registerNewMint(uint256 tokenId)
    external
    authorizedFor(Permission.RegisterNewMints)
{
    require(dividends[tokenId] == 0, "Token already registered");

    if (totalDividend == 0) {
        dividends[tokenId] = 1; // to distinguish EpicHeroNFT added before the first reflection from unregistered EpicHeroNFT
    } else {
        dividends[tokenId] = totalDividend;
    }

    totalEpicHero++;
}
```

❖ The owner can register a dividend token.

```
ftrace | funcSig
function batchRegisterMints(uint256[] memory tokenIds)
    external
    authorizedFor(Permission.RegisterNewMints)
{
    bool hasDuplicate;
    for (uint256 i = 0; i < tokenIds.length; i++) {
        uint256 tokenId = tokenIds[i];

        if (dividends[tokenId] != 0) {
            hasDuplicate = true;
            break;
        }

        if (totalDividend == 0) {
            dividends[tokenId] = 1; // to distinguish EpicHeroNFT added before the first reflection from unregistered EpicHeroNFT
        } else {
            dividends[tokenId] = totalDividend;
        }
    }

    require(!hasDuplicate, "Contains already registered token(s)");

    totalEpicHero += tokenIds.length;
}
```

❖ The owner can change the WBNB address.

```
ftrace | funcSig
function setWbnbAddress(address _newAddress)
    external
    authorizedFor(Permission.AdjustVariables)
{
    wbnbAddress = _newAddress;
}
```

❖ The owner can change the epic hero NFT address.

```
ftrace | funcSig
function setEpicHeroNFTAddress(address _newAddress)
    external
    authorizedFor(Permission.AdjustVariables)
{
    epicHeroNFTAddress = _newAddress;
    EpicHeroNFT = IEpicHeroNFT(_newAddress);
}
```

❖ The owner can manually set dividend attributes.

```
// use only in emergencies, could lead to some people being unable
ftrace | funcSig
function teamSetDividend(
    uint256 tokenId,
    uint128 lastDividendAt,
    uint128 realized
) external authorizedFor(Permission.AdjustVariables) {
    _setDividend(tokenId, lastDividendAt, realized);
}
```

❖ The owner can retrieve the dividend token balance to the owner's wallet.

```
ftrace | funcSig
function retrieveTokens(address token, uint256 amount)
    external
    authorizedFor(Permission.RetrieveTokens)
{
    uint256 balance = IBEP20(token).balanceOf(address(this));

    if (amount > balance) {
        amount = balance;
    }

    require(IBEP20(token).transfer(msg.sender, amount), "Transfer failed");
}
```

❖ The owner can retrieve the dividend BNB balance to owner's wallet.

```
ftrace | funcSig
function retrieveBNB(uint256 amount)
    external
    authorizedFor(Permission.RetrieveTokens)
{
    uint256 balance = address(this).balance;

    if (amount > balance) {
        amount = balance;
    }

    (bool success, ) = payable(msg.sender).call{value: amount}("");
    require(success, "Failed");
}
```

# Audit conclusion

While conducting the audit of the Epic Hero Reflection smart contract, it was observed that there is nothing alarming with the code.