RugFreeCoins Audit

Epic Hero 3D NFT Token

Smart Contract Security Audit

September 15, 2021

Contents

Audit details

Audited project

Epic Hero 3D NFT Token

Contract Address

0xafDcB0eCaD1c8Cb22893dCA7D6c510dBFDa3BBeC

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/token/0xafDcB0eCaD1c8Cb22893dCA7D6c510dBFDa3BBeC

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

Epic Hero 3D NFT is the contract where the fee of 0.25% when buying and 0.75% when selling from the main contract of Epic Hero getting sent for this contract to distribute among the game winners and activities related to the game.

Contract details

Token contract details for 15th September 2021

Contract name	Epic Hero 3D NFT
Contract address	0xafDcB0eCaD1c8Cb22893dCA7D6c510dBFDa3BBeC
Token ticker	EpicHero3DNFT
Epic Hero address	0x47cc5334f65611ea6be9e933c49485c88c17f5f0
Fee receiver	0x9bf2891fa94f6d9954d6eafa89c759b3c9ddc05a
Reflect address	0x09eaf2a4bce29796ee380aae6a3d23b817ad67eb
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		
L	Function Name	Visibility	Mutability	Modifiers
IBEP20	**Interface**			
L	totalSupply	External ❗		NO❗
L	balanceOf	External ❗		NO❗
L	transfer	External ❗	🛑	NO❗
L	allowance	External ❗		NO❗
L	approve	External ❗	🛑	NO❗
L	transferFrom	External ❗	🛑	NO❗
IERC165	**Interface**			
L	supportsInterface	External ❗		NO❗
IERC721	**Interface**	**IERC165**		
L	balanceOf	External ❗		NO❗
L	ownerOf	External ❗		NO❗

L	safeTransferFrom	External ❗	●	NO❗
L	transferFrom	External ❗	●	NO❗
L	approve	External ❗	●	NO❗
L	getApproved	External ❗		NO❗
L	setApprovalForAll	External ❗	●	NO❗
L	isApprovedForAll	External ❗		NO❗
L	safeTransferFrom	External ❗	●	NO❗
IERC721Receiver	**Interface**			
L	onERC721Received	External ❗	●	NO❗
IERC721Metadata	**Interface**	**IERC721**		
L	name	External ❗		NO❗
L	symbol	External ❗		NO❗
L	tokenURI	External ❗		NO❗
Address	**Library**			
L	isContract	Internal 🔒		
Strings	**Library**			

L	toString	Internal 🔒		
Math	**Library**			
L	max	Internal 🔒		
L	min	Internal 🔒		
L	average	Internal 🔒		
Context	**Implementation**			
L	_msgSender	Internal 🔒		
L	_msgData	Internal 🔒		
ERC165	**Implementation**	**IERC165**		
L	supportsInterface	Public ❗		NO❗
ERC721	**Implementation**	**Context, ERC165, IERC721, IERC721Metadata**		
L		Public ❗	🛑	NO❗
L	supportsInterface	Public ❗		NO❗
L	balanceOf	Public ❗		NO❗
L	ownerOf	Public ❗		NO❗

L	name	Public ❗		NO❗
L	symbol	Public ❗		NO❗
L	tokenURI	Public ❗		NO❗
L	_baseURI	Internal 🔒		
L	approve	Public ❗	🛑	NO❗
L	getApproved	Public ❗		NO❗
L	setApprovalForAll	Public ❗	🛑	NO❗
L	isApprovedForAll	Public ❗		NO❗
L	transferFrom	Public ❗	🛑	NO❗
L	safeTransferFrom	Public ❗	🛑	NO❗
L	safeTransferFrom	Public ❗	🛑	NO❗
L	_safeTransfer	Internal 🔒	🛑	
L	_exists	Internal 🔒		
L	_isApprovedOrOwner	Internal 🔒		
L	_safeMint	Internal 🔒	🛑	
L	_safeMint	Internal 🔒	🛑	
L	_mint	Internal 🔒	🛑	
L	_burn	Internal 🔒	🛑	

L	_transfer	Internal 🔒	🛑	
L	_approve	Internal 🔒	🛑	
L	_checkOnERC721Received	Private 🔐	🛑	
L	_beforeTokenTransfer	Internal 🔒	🛑	
IERC721Enumerable	**Interface**	**IERC721**		
L	totalSupply	External ❗		NO❗
L	tokenOfOwnerByIndex	External ❗		NO❗
L	tokenByIndex	External ❗		NO❗
IEpicHeroReflect	**Interface**			
L	registerNewMint	External ❗	🛑	NO❗
L	updateBurnedToken	External ❗	🛑	NO❗
ERC721Enumerable	**Implementation**	**ERC721, IERC721Enumerable**		
L	supportsInterface	Public ❗		NO❗
L	tokenOfOwnerByIndex	Public ❗		NO❗
L	totalSupply	Public ❗		NO❗
L	tokenByIndex	Public ❗		NO❗

L	_beforeTokenTransfer	Internal 🔒	◉	
L	_addTokenToOwnerEnumeration	Private 🔐	◉	
L	_removeTokenFromOwnerEnumeration	Private 🔐	◉	
EpicAuth	**Implementation**			
L		Public ❗	◉	NO❗
L	authorizedFor	Internal 🔒		
L	authorizeFor	Public ❗	◉	NO❗
L	authorizeForMultiplePermissions	Public ❗	◉	NO❗
L	unauthorizeFor	Public ❗	◉	NO❗
L	unauthorizeForMultiplePermissions	Public ❗	◉	NO❗
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 ❗	🛑	NO❗
L	unlockPermission	Public ❗	🛑	NO❗
EpicHeroNFT	**Implementation**	**ERC721Enumerable, EpicAuth**		
L		Public ❗	🛑	ERC721 EpicAuth
L	_baseURI	Internal 🔒		
L	purchasePack	External ❗	🛑	NO❗
L	_mintCardsOfPack	Internal 🔒	🛑	
L	levelUp	External ❗	🛑	NO❗
L	getHero	External ❗		NO❗
L	getPrice	Public ❗		NO❗
L	getPacks	External ❗		NO❗
L	getLevelUpPrices	External ❗		NO❗
L		External ❗	💵	NO❗
L	setBaseURI	External ❗	🛑	NO❗
L	setThoreumAddress	External ❗	🛑	NO❗
L	setEpicHeroAddress	External ❗	🛑	NO❗
L	setReflectAddress	External ❗	🛑	NO❗

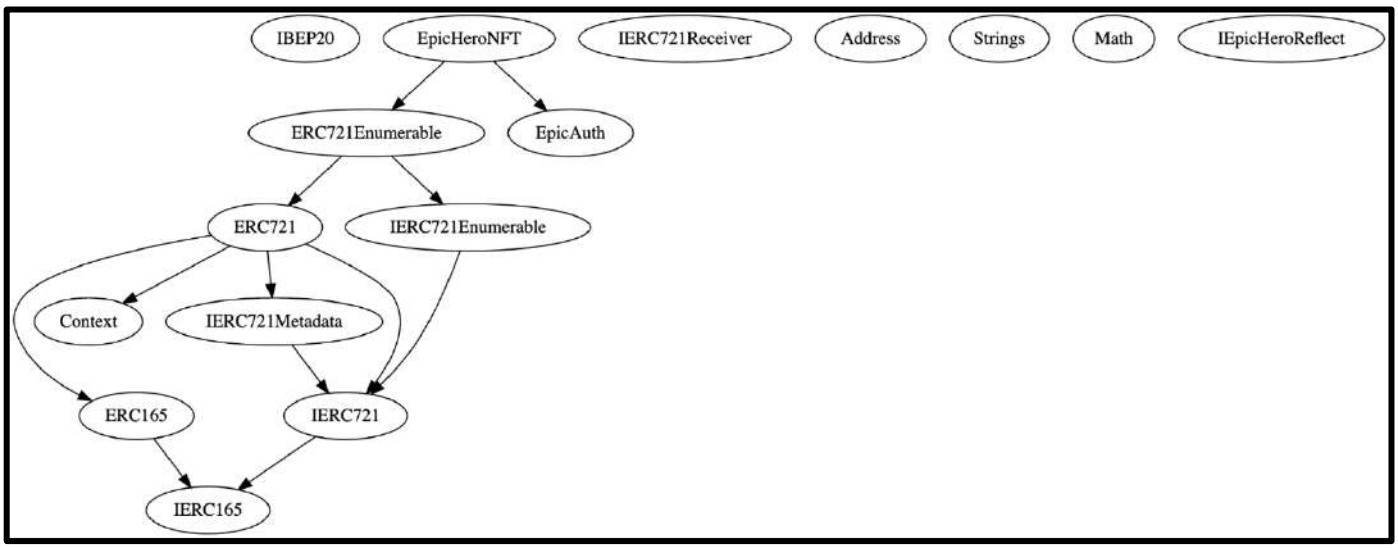
L	setMaxLevel	External !	●	NO!
L	setMaxRarity	External !	●	NO!
L	addPack	Public !	●	NO!
L	editPack	External !	●	NO!
L	setSaleRunning	Public !	●	NO!
L	addCardSet	Public !	●	NO!
L	editCardSet	Public !	●	NO!
L	addAttribute	Public !	●	NO!
L	editAttribute	Public !	●	NO!
L	addLevelUpPrice	Public !	●	NO!
L	editLevelUpPrice	Public !	●	NO!
L	compareStrings	Internal 🔒		
L	adminSetAttribute	Public !	●	NO!
L	adminMintPack	External !	●	NO!
L	adminMintSingle	External !	●	NO!
L	adminMintMultiple	External !	●	NO!
L	adminKillHero	External !	●	NO!
L	adminSetLevel	External !	●	NO!

L	adminSetRarity	External !		NO!
L	retrieveTokens	External !		NO!
L	retrieveBNB	External !		NO!
L	_createHero	Internal		
L	_singleMint	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 the wallet.

```
/**
 * 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 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 unauthorize permissions for wallets.

```
/**
 * Remove address' authorization
 */
ftrace | funcSig
function unauthorizeFor(address adr, string memory permissionName) public {
    authorizedFor(Permission.Unauthorize);
    require(adr != owner, "!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 change the base URL.

```
ftrace | funcSig
function setBaseURI(string memory baseURI_) external {
    authorizedFor(Permission.AdjustVariables);

    _baseUriExtended = baseURI_;
}
```

❖ The owner can change Thoreum, Epic hero and reflect address.

```
ftrace | funcSig
function setThoreumAddress(address _newAdr) external {
    authorizedFor(Permission.AdjustVariables);

    thoreumAddress = _newAdr;
    ThoreumToken = IBEP20(_newAdr);
}

ftrace | funcSig
function setEpicHeroAddress(address _newAdr) external {
    authorizedFor(Permission.AdjustVariables);

    epicHeroAddress = _newAdr;
    EpicHeroToken = IBEP20(_newAdr);
}

ftrace | funcSig
function setReflectAddress(address _newAdr) external {
    authorizedFor(Permission.AdjustVariables);

    reflectAddress = _newAdr;
}
```

❖ The owner can change max level and max rarity.

```
ftrace | funcSig
function setMaxLevel(uint8 newMaxLevel) external {
    authorizedFor(Permission.AdjustVariables);

    require(newMaxLevel > maxLevel);
    maxLevel = newMaxLevel;
}

ftrace | funcSig
function setMaxRarity(uint8 newMaxRarity) external {
    authorizedFor(Permission.AdjustVariables);

    require(newMaxRarity > maxRarity);
    maxRarity = newMaxRarity;
}
```

❖ The owner can add a new pack.

```
ftrace | funcSig
function addPack(
    uint232 basePrice,
    uint8 numberOfCards,
    bool saleRunning,
    uint8 cardSetId,
    address tokenAddress
) public {
    authorizedFor(Permission.ManagePacks);

    packTypes.push(
        Pack(basePrice, numberOfCards, saleRunning, cardSetId, tokenAddress)
    );

    emit PackAdded(
        packTypes.length - 1,
        basePrice,
        numberOfCards,
        cardSetId,
        tokenAddress
    );
}
```

❖ The owner can edit the pack.

```
ftrace | funcSig
function editPack(
    uint8 packId,
    uint232 basePrice,
    uint8 numberOfCards,
    bool saleRunning,
    uint8 cardSetId,
    address tokenAddress
) external {
    authorizedFor(Permission.ManagePacks);

    packTypes[packId].basePrice = basePrice;
    packTypes[packId].numberOfCards = numberOfCards;
    packTypes[packId].saleRunning = saleRunning;
    packTypes[packId].cardSetId = cardSetId;
    packTypes[packId].tokenAddress = tokenAddress;

    emit PackEdited(
        packId,
        basePrice,
        numberOfCards,
        saleRunning,
        cardSetId,
        tokenAddress
    );
}
```

❖ The owner can enable/disable sales in packs.

```
ftrace | funcSig
function setSaleRunning(uint256 packId, bool running) public {
    authorizedFor(Permission.ManagePacks);

    packTypes[packId].saleRunning = running;
}
```

❖ The owner can add and edit card sets.

```
ftrace | funcSig
function addCardSet(uint64 mintLimit) public {
    authorizedFor(Permission.ManagePacks);

    cardSets.push(CardSet(0, mintLimit));

    emit CardSetAdded(cardSets.length - 1, mintLimit);
}

ftrace | funcSig
function editCardSet(uint8 setId, uint64 mintLimit) public {
    authorizedFor(Permission.ManagePacks);

    cardSets[setId].mintLimit = mintLimit;

    emit CardSetEdited(setId, mintLimit);
}
```

❖ The owner can add new attributes.

```
ftrace | funcSig
function addAttribute(string memory name) public {
    authorizedFor(Permission.ManageAttributes);

    attributes.push(Attribute(name));
    attributeIndex[name] = attributes.length - 1;
    attributeExists[name] = true;

    emit AttributeAdded(attributes.length - 1, name);
}
```

❖ The owner can edit attributes.

```
ftrace | funcSig
function editAttribute(uint8 attrId, string memory name) public {
    authorizedFor(Permission.ManageAttributes);

    Attribute memory old = attributes[attrId];

    if (compareStrings(old.name, name) == false) {
        delete attributeIndex[old.name];
        attributeIndex[name] = attrId;

        delete attributeExists[old.name];
        attributeExists[name] = true;
    }

    attributes[attrId].name = name;

    emit AttributeEdited(attrId, name);
}
```

❖ The owner can add a level up price.

```
ftrace | funcSig
function addLevelUpPrice(uint256 _thoreum, uint256 _epicHero) public {
    authorizedFor(Permission.ManageAttributes);

    levelUpPrices.push(LevelUpPrice(_thoreum, _epicHero));
}
```

❖ The owner can edit level up price.

```
ftrace | funcSig
function editLevelUpPrice(
    uint8 level,
    uint256 _thoreum,
    uint256 _epicHero
) public {
    authorizedFor(Permission.ManageAttributes);

    require(level < levelUpPrices.length, "Invalid level");

    levelUpPrices[level].thoreum = _thoreum;
    levelUpPrices[level].epicHero = _epicHero;
}
```

- The owner can mint new packs.

```
ftrace | funcSig
function adminMintPack(uint8 packId, address recipient) external {
    authorizedFor(Permission.Mint);

    if (recipient == address(0)) recipient = msg.sender;
    Pack memory pack = packTypes[packId];

    cardSets[pack.cardSetId].minted += pack.numberOfCards;
    _mintCardsOfPack(recipient, packId, pack.numberOfCards);
}
```

- The owner can mint single pack.

```
ftrace | funcSig
function adminMintSingle(
    uint8 packId,
    uint8 level,
    uint8 rarity,
    address recipient
) external {
    authorizedFor(Permission.Mint);

    if (recipient == address(0)) recipient = msg.sender;

    _singleMint(recipient, packId, level, rarity);
}
```

- The owner can kill heros.

```
ftrace | funcSig
function adminKillHero(uint256 heroId) external {
    authorizedFor(Permission.Mint);

    safeTransferFrom(ownerOf(heroId), deadAddress, heroId);

    try
        IEpicHeroReflect(reflectAddress).updateBurnedToken(heroId)
    {} catch {}

    emit AdminKillHero(heroId);
}
```

❖ The owner can set the level to the hero.

```
ftrace | funcSig
function adminSetLevel(uint256 heroId, uint8 newLevel) external {
    authorizedFor(Permission.ManageAttributes);

    require(newLevel <= maxLevel, "Max level");
    Hero storage hero = _heroes[heroId];
    hero.level = newLevel;

    if (newLevel == 1) {
        try
            IEpicHeroReflect(reflectAddress).registerNewMint(heroId)
        {} catch {}
    }

    emit AdminSetLevel(heroId, newLevel);
}
```

❖ The owner can set rarity to the hero.

```
ftrace | funcSig
function adminSetRarity(uint256 heroId, uint8 rarity) external {
    authorizedFor(Permission.ManageAttributes);

    Hero storage hero = _heroes[heroId];

    require(rarity <= maxRarity, "Max rarity");

    hero.rarity = rarity;
}
```

❖ The owner can retrieve tokens and BNB balance to the owner account.

```
ftrace | funcSig
function retrieveTokens(address token, uint256 amount) external {
    authorizedFor(Permission.Withdraw);

    uint256 balance = IBEP20(token).balanceOf(address(this));

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

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

ftrace | funcSig
function retrieveBNB(uint256 amount) external {
    authorizedFor(Permission.Withdraw);

    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 3D NFT smart contract, it was observed that there is nothing alarming with the code.