PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Bulls Inc NFT

08 February 2022

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared Bulls Inc NFT on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Bulls Inc NFT
URL	https://bulls.inc/
Platform	Ethereum
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`BullsIncNFT`	`0x31c1fb9901dc6ef9c6cacd49c3bcd79fe2486e6c`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	0	-	-	-
Low	3	3	-	-
Informational	5	-	-	5
Total	**11**	**6**	**-**	**5**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 BullsIncNFT [[Bulls.inc [BINC]]

ID	Severity	Summary	Status
01	HIGH	Reentrancy risk in `mint` method	RESOLVED
02	HIGH	`burn` will cause the minting to stop	RESOLVED
03	HIGH	The whole supply can not be reached if `mint_promo` is called	RESOLVED
04	LOW	`mint_promo` does not adhere to checks-effects-interactions pattern	RESOLVED
05	LOW	A contract can mint all the supply if caller is a non-EOA	RESOLVED
06	LOW	Extra ETH not refunded automatically	RESOLVED
07	INFO	Gas Optimization: `maxSupply` and `maxPromoSupply` can be made constant	ACKNOWLEDGED
08	INFO	Gas Optimization: `teamAddresses` should be transformed into a map	ACKNOWLEDGED
09	INFO	Several variables can be made `external`	ACKNOWLEDGED
10	INFO	Lack of events for several functions	ACKNOWLEDGED
11	INFO	Ambiguous error on `burn`	ACKNOWLEDGED

2 Findings

2.1 BullsIncNFT [[Bulls.inc [BINC]]

BullsIncNFT_flat is an ERC721 NFT token that implements ERC721Enumerable pattern which is a known pattern in the industry, used to get different data about the minted tokens. The Bulls.Inc maximum supply is 4444, which includes a promo supply of 100 NFTs that are reserved for and minted by the team.

There are 2 phases of the minting process of the Bulls.Inc. The whitelist phase (which is determined by the variable `whitelistPhase`) which will give to anyone the possibility to mint up to `maxMintAmount` tokens (currently set as 10, this can be changed by the owner) and second phase when whitelist is disabled, users can mint as many tokens as they want, and the minting process is capped by the `maxSupply` of 4444.

Every mint will cost the investor a value determined by the `cost` variable (currently set as 0.1 ETH per token — this can be changed by the owner).

❗ Additionally, the owner can mint at any time as many tokens as he wants from the main batch of 4444 without paying ETH by calling the `mint` method.

2.1.1 Privileges

The following functions can be called by the owner of the contract:

- `renounceOwnership`
- `transferOwnership`
- `mint_promo`
- `reveal`
- `setCost`
- `setMaxMintAmount`
- `setNotRevealedURI`
- `setBaseURI`
- `setBasePromoURI`
- `setBaseExtension`
- `setTeamAddresses`
- `setWhitelistPhase`
- `pause`
- `withdraw`

2.1.2 Issues & Recommendations

Issue #01	Reentrancy risk in `mint` method
Severity	HIGH SEVERITY
Description	The `mint` method is used by the investors to mint tokens for ETH. Every address has a maximum number of mints it can do (default is 10) — after each `safeMint`, the `addressToMintedAmount` variable is updated for `msg.sender` with the `_mintAmount` (a variable sent by the user to determine how many NFTs they wants to mint). This goes against the checks-effects-interactions pattern due to the fact that the update of the `addressToMintedAmount` is done after the `safeMint` trigger. The `safeMint` method behavior is as follows: after each mint, it checks if the receiver of the NFT is a valid one (an EOA or a smart contract that implements an interface called `onERC721Received`). This makes it possible for an attacker to call the `mint` method again inside the `onERC721Received` method which will allow them to mint another batch before updating `addressToMintedAmount[msg.sender]`.
Recommendation	Consider moving the update of `addressToMintedAmount[msg.sender]` before the `safeMint`.
Resolution	RESOLVED

Issue #02	**burn will cause the minting to stop**
Severity	HIGH SEVERITY
Description	The `mint` method is used by the investors to mint tokens for ETH. Inside this method, the `totalSupply()` method is used to determine the next id that needs to be minted. However, as the burn functionality is implemented inside the BullsInc NFT contract, if someone calls `burn` before the minting is finished, it will cause the minting to revert. Example Mint token 1, 2 and 3 then burn token 2 will cause the minting to stop as the next id will be 3 which already exists. ❗ Additionally once the minting is completed, the minting functionality can theoretically be re-enabled if someone burns a token.
Recommendation	Paladin recommends using a `nextId` variable inside the contract that stores the `nextId` available to mint. ❗ We also recommend either adding an extra check where `require(nextId <= maxSupply, "Max supply reached")` or calling the pause method once the supply is minted.
Resolution	RESOLVED

Issue #01	The whole supply can not be reached if `mint_promo` is called
Severity	HIGH SEVERITY
Description	The `mint` method is used by the investors to mint tokens for ETH. The `mint_promo` method is used by `onlyOwner` to mint promo NFTs that are going above the 4444 supply, currently capped at 100. Due to the fact that the `mint` method uses the `totalSupply()` to calculate the next ids available for mint, the last tokens will not be available to mint if `mint_promo` is called before reaching the initial supply of the 4444, instead they will revert with the token already exists. Example: Users mint the first 1000 tokens, then `onlyOwner` mints 100 promo tokens using `mint_promo`. When the supply reaches 4344, the minting will stop because the `totalSupply` will count both the tokens minted by the users and the ones minted by the owner, leaving the last 100 tokens unminted.
Recommendation	Consider taking into account the supply minted by the mint_promo in the mint method.
Resolution	RESOLVED The 100 NFTs that are reserved for the owner are now included in the 4444 max supply.

Issue #04	`mint_promo` does not adhere to checks-effects-interactions pattern
Severity	LOW SEVERITY
Description	The `mint_promo` method is used by the team to mint tokens without paying any ETH. The Bulls Inc team has set a `maxPromoSupply` (currently 100) which is checked against `promoSupply + _mintAmount`. Unfortunately, the checks-effects-interactions pattern is not adhered to due to the fact that the update of the `promoSupply` is done after the `safeMint` trigger. We would like to note that this is not a threat like the the `mint` method reentrancy issue due to the fact that the token id is created from `maxSupply + promoSupply + i`, which causes the reentrancy call to fail because it will try to mint a token that already exists.
Recommendation	Consider moving the update of `promoSupply` before the `safeMint`.
Resolution	RESOLVED

Issue #05	A contract can mint all the supply if caller is a non-EOA
Severity	LOW SEVERITY
Description	Inside the `mint` function, the requirement for `require(_mintAmount <= maxMintAmount, "Exceeds mint limit." );` can be skipped if the calls are done by a chain of subcontracts. This happened also when Adidas released their NFT project. https://twitter.com/Montana_Wong/status/1472023753865396227
Recommendation	Consider restricting the call to the mint method to strictly being an EOA: `require(tx.origin==msg.sender, "Not an EOA");`
Resolution	RESOLVED

Issue #06	Extra ETH not refunded automatically
Severity	LOW SEVERITY
Location	Line 1283 `require(msg.value >= cost * _mintAmount, "Not enough value sent.");`
Description	Inside the `mint` function, the requirement for the amount of ETH sent for buying the token checks if someone sent at least the `cost * _mintAmount` of ETH, meaning that if someone sends more than the required amount, he would not be refunded automatically.
Recommendation	Consider restricting the requirement to `require(msg.value == cost * _mintAmount, "Not enough value sent.");`
Resolution	RESOLVED

Issue #07	Gas Optimization: `maxSupply` and `maxPromoSupply` can be made constant
Severity	INFORMATIONAL
Description	Variables that are never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly constant.
Resolution	ACKNOWLEDGED

Issue #08	Gas Optimization: `teamAddresses` should be transformed into a map
Severity	INFORMATIONAL
Location	Line 1225-1234

```
bool team = false;
for (uint x = 0; x<teamAddresses.length;x++){
    if (teamAddresses[x]==msg.sender){
        team = true;
    }
}
require(team, "Only Team");
_;
```

Description	The `teamAddresses` variable is used to keep a list of team wallets that can claim promo tokens. This variable is currently an array that is iterated all the time when calling `onlyTeam` modifier. Iterating through an array to check if an address exists can significantly increase the gast cost, mainly as this is called whenever the `mint_promo` method is triggered.
Recommendation	In order to save gas, Paladin recommends transforming the `teamAddresses` into a mapping variable `mapping(address => bool) teamAddresses`; and in the modifier, to have just a simple require `require(teamAddress[msg.sender], "Only Team");` This will save gas whenever `mint_promo` is called and also saves gas when deploying as the modifier is lower in code size.
Resolution	ACKNOWLEDGED

Issue #09	Several variables can be made `external`
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the `external` keyword. - `mint` - `mint_promo` - `burn` - `reveal` - `setCost` - `setMaxMintAmount` - `setNotRevealedURI` - `setBaseURI` - `setBasePromoURI` - `setBaseExtension` - `setTeamAddresses` - `setWhitelistPhase` - `pause` - `withdraw` Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above functions as `external`.
Resolution	ACKNOWLEDGED

Issue #10	Lack of events for several functions
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications: - `reveal` - `setCost` - `setMaxMintAmount` - `setNotRevealedURI` - `setBaseURI` - `setBasePromoURI` - `setBaseExtension` - `setTeamAddresses` - `setWhitelistPhase` - `pause` - `withdraw`
Recommendation	Add events for the above functions.
Resolution	ACKNOWLEDGED

Issue #11	Ambiguous error on `burn`
Severity	INFORMATIONAL
Description	The `burn` method is used to burn a token by the owner or an approved wallet. The `require` check used for this does not revert with an explicit message.
Recommendation	Consider adding a message for the `require` check.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY