PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Trader Joe (LaunchPeg)

24 May 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 for Trader Joe's LaunchPeg contracts on the Avalanche 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	Trader Joe (LaunchPeg)
URL	https://traderjoexyz.com
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
LaunchpegFactory	Proxy: 0x7BFd7192E76D950832c77BB412aaE841049D8D9B Implementation: 0x454206AD825cAfaE03c9581014AF6b74f7D53713	MATCH
BaseLaunchPeg	Dependency of FlatLaunchPeg and Launchpeg	MATCH
FlatLaunchPeg	Implementation (used by all FlatLaunchPeg deployments): 0xB7E39B647b1ddcA45d58b9fb1F265C7c25627e6F	MATCH
Launchpeg	Implementation (used by all Launchpeg deployments): 0xB22c32B0bF1a6A7c50f0c528c3566E7dB12659D4	MATCH
BatchReveal	This component was modified to include Chainlink VRF logic after the audit. This logic was not audited and users should be extremely careful with NFTs that use it.	FAIL
ERC721A	Dependency of BaseLaunchPeg	MATCH
LaunchPegErrors	Dependency of all contracts	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	2	-	-
Medium	3	3	-	-
Low	5	2	1	2
Informational	15	14	-	1
Total	**25**	**21**	**1**	**3**

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 LaunchpegFactory

ID	Severity	Summary	Status
01	INFO	Typographical errors	✓ RESOLVED

1.3.2 BaseLaunchPeg

ID	Severity	Summary	Status
02	HIGH	`initializeJoeFee` can be called multiple times, allowing the creator to take the fee that is supposed to go to Trader Joe	✓ RESOLVED
03	MEDIUM	`devMint` lacks collection size validation	✓ RESOLVED
04	MEDIUM	Missing safeguards on royalty information	✓ RESOLVED
05	LOW	Changing of URL should be locked after all NFTs have been minted and revealed	ACKNOWLEDGED
06	LOW	`_amountMintedByDevs` is private	✓ RESOLVED
07	INFO	Gas optimizations	✓ RESOLVED
08	INFO	Inherited `init` functions not called in the base contract	✓ RESOLVED
09	INFO	Lack of validation	ACKNOWLEDGED
10	INFO	Typographical error	✓ RESOLVED
11	INFO	`withdrawAvax` should have a `to` parameter in case the owner does not have a fallback function	✓ RESOLVED
12	INFO	`supportsInterface` does not indicate that it supports all interfaces	✓ RESOLVED

1.3.3 FlatLaunchPeg

ID	Severity	Summary	Status
13	HIGH	Insufficient check for `maxPerAddressDuringMint` during `publicSaleMint`	✓ RESOLVED
14	INFO	Irregularities and unexpected behaviour during `Mint` events	✓ RESOLVED
15	INFO	Lack of validation	✓ RESOLVED
16	INFO	Duplicate import	✓ RESOLVED
17	INFO	Gas optimizations	✓ RESOLVED

1.3.4 LaunchPeg

ID	Severity	Summary	Status
18	LOW	Missing safeguards on `initializePhases`	✓ RESOLVED
19	INFO	Lack of validation	✓ RESOLVED
20	INFO	Duplicate imports	✓ RESOLVED
21	INFO	Unexpected behaviour during `Mint` events in case of reentrancy	✓ RESOLVED

1.3.5 BatchReveal

ID	Severity	Summary	Status
22	MEDIUM	`_getShuffledTokenId` can run out of gas	✓ RESOLVED
23	LOW	Last incomplete batch cannot be revealed by users	PARTIAL
24	LOW	Tokenomics: Premature revealing could incentivize waiting behaviour	ACKNOWLEDGED

1.3.6 ERC721A

Small informational findings were found while auditing the ERC721A dependency, but they have been omitted from the report as resolving them would provide no value to the JoePeg system and this is considered an external dependency which we recommend leaving unchanged.

1.3.7 LaunchPegErrors

ID	Severity	Summary	Status
25	INFO	Unused functionality	✓ RESOLVED

2 Findings

2.1 LaunchpegFactory

`LaunchpegFactory` is the main interface for users to create new JoePeg NFTs. It has two main user facing functions: `createFlatLaunchpeg` and `createLaunchPeg`.

Firstly, it allows users to create a "flat" launchpeg NFT which is the most straightforward way of distributing an NFT within the JoePeg system. A flat launchpeg simply allows the creator to set an allowlist price, public sale price and the allowlist allowances for a set of users they would like to whitelist. More on this NFT type can be read in its respective section in this audit.

Secondly, it allows users to create a general launchpeg NFT. This extends the flat launchpeg with auction functionality for the initial distribution. More can be read about this NFT type in its respective section within this report.

Whenever users create an NFT through the factory, they need to specify a variety of parameters: the name, symbol, owner, `royaltyReceiver` (artist), collection size, amount for sale and so forth.

Technically speaking, the factory uses the Clones pattern which means that Trader Joe will always deploy two reference implementations for both NFT types. All NFTs deployed through the factory are then a thin minimal proxy pointing to these reference implementations. This pattern is generally considered secure and is desired because it significantly decreases the deployment cost in terms of gas due to the tiny size of a minimal proxy. More about the advantages of minimal proxies can be read in EIP-1167 (https://eips.ethereum.org/EIPS/eip-1167).

It should be noted that Trader Joe can update the implementations for both NFTs. This will not affect any deployed NFTs but will affect NFTs that are deployed after the update.

The Factory defines the fee that will be levied and transferred to Trader Joe on any NFT sale. This fee can be changed freely by Trader Joe but we expect them to clearly disclose this on the frontend. Once an NFT launch is deployed, the change of this fee is no longer forwarded to the launch which means that the fee is locked in at the time of deployment.

`LaunchpegFactory` is upgradeable which means that Trader Joe can adjust the behaviour of the contract after the audit to anything they please. Users should carefully evaluate that the deployed implementation matches the live matched contract in this audit.

2.1.1 Privileges

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

- `setLaunchpegImplementation`
- `setFlatLaunchpegImplementation`
- `setDefaultJoeFeePercent`
- `setDefaultJoeFeeCollector`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	Typographical errors
Severity	INFORMATIONAL
Location	Line 73 `function numLaunchpegs(uint256 _launchegType)`
Description	The parameter within this function should be `_launchpegType`, not `_launchegType`.
Recommendation	Consider fixing the typographical error.
Resolution	RESOLVED

2.2 BaseLaunchPeg

`BaseLaunchpeg` is an abstract contract that contains the core NFT functionality of a `LaunchPeg` sale. The contract follows the minimal proxy pattern which is a well known pattern that allows for the contract to be extremely cheap to deploy as the logic is actually contained in a globally shared "implementation" contract.

The contract implements the ERC-2981 royalty standard. This standard allows contracts, such as NFTs that support ERC-721 and ERC-1155 interfaces, to signal a royalty amount to be paid to the NFT creator or rights holder every time the NFT is sold or re-sold. This is intended for NFT marketplaces that want to support the ongoing funding of artists and other NFT creators. The contract uses the reference OpenZeppelin ERC-2981 implementation to support this functionality. The default royalty fee is 5%.

This base contract is inherited within `LaunchPeg` and `FlatLaunchPeg`. It will generally not be deployed by itself.

For the core NFT functionality, `BaseLaunchpeg` does not extend the reference OpenZeppelin NFT implementation; instead, it extends ERC721A, which is included within the scope of this audit and was developed by Chiru Labs. More about the ERC721A implementation can be read within its section in this report, but in summary, it essentially uses some clever tricks to reduce the gas costs of minting large amounts of tokens in a single transaction. Normally, each minted token would cost additional gas within ERC721; however, the gas cost is fixed regardless of the mint amount.

The contract does not support NFTs with a total supply greater than 2**64, which as far as we know do not exist.

2.2.1 Privileges

The following functions can be called by the owner and other privileged roles of the contract:

- `devMint (onlyProjectOwner)`
- `forceReveal (onlyProjectOwner)`
- `initializeJoeFee`
- `setRoyaltyInfo`
- `seedAllowlist`
- `setBaseURI`
- `setUnrevealedURI`
- `setProjectOwner`
- `withdrawAvax`
- `renounceOwnership`
- `transferOwnership`

2.2.2 Issues & Recommendations

Issue #02	**`initializeJoeFee` can be called multiple times, allowing the creator to take the fee that is supposed to go to Trader Joe**
Severity	HIGH SEVERITY
Description	The `BaseLaunchpeg` contract sets a `joeFeePercent` that represents a fee taken by TraderJoe when the owner withdraws the AVAX from the contract. Currently, the method to configure this fee on a sale can be called multiple times. This allows the creator to set themselves as the "Trader Joe fee recipient" to receive Trader Joe's share of the fees.
Recommendation	Consider making the function callable only once.
Resolution	RESOLVED The function can only be initialised once now.

Issue #03	**`devMint` lacks collection size validation**
Severity	MEDIUM SEVERITY
Description	Presently the `devMint` function does not check whether the collection size has been reached. This theoretically allows the contract to exceed the collection size.
Recommendation	Consider validating that the total amount minted is still smaller than the collection size including the `_quantity` of this function.
Resolution	RESOLVED `devMint` can now no longer mint in excess of the collection size.

Issue #04	Missing safeguards on royalty information
Severity	MEDIUM SEVERITY
Description	The BaseLaunchpeg contract implements ERC 2981 standard. This standard allows contracts, such as NFTs that support ERC-721 and ERC-1155 interfaces, to signal a royalty amount to be paid to the NFT creator or rights holder every time the NFT is sold or re-sold. This is intended for NFT marketplaces that want to support the ongoing funding of artists and other NFT creators. The privileged `setRoyaltyInfo` function sets the royalty fee and royalty receiver (who should receive the fee when a trade is happening on the marketplace) information for every NFT that inherits the BaseLaunchpeg contract. Currently, there are no safeguards that can stop the owner from setting the royalty fee to 100%. Even though this does not affect the contract itself, it can be a problem when the NFTs are released on different marketplaces that support this standard. The marketplaces will retrieve the fee from the `royaltyInfo` function and if they set the variable to 100%, the full amount paid will be transferred to the royalty receiver and not to the seller if a trade happens on that marketplace.
Recommendation	Consider adding a cap to the royalty fee.
Resolution	RESOLVED The fee can now be set to a maximum of 25%.

Issue #05	**Changing of URL should be locked after all NFTs have been minted and revealed**
Severity	LOW SEVERITY
Description	As NFTs are non-fungible tokens, after all NFTs have been minted and revealed, the changing of `baseURL` should be locked to preserve the non-fungible state of the token.
Recommendation	Consider introducing a locking mechanism on changing `baseURL` that can be triggered once everything is revealed and minted.
Resolution	ACKNOWLEDGED The client wants to have the ability to change the URL in case of any issue with IPFS.

Issue #06	**`_amountMintedByDevs` is `private`**
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the variable as public.
Resolution	RESOLVED

Issue #07	Gas optimizations
Severity	INFORMATIONAL

Description	Throughout the contract we discovered some gas optimizations that can be applied:
	- Within the `initializeBaseLaunchpeg` function, the `_name` and `_symbol` parameters can be declared as `calldata` to save gas.
	- Within the sendAllowlist function, _addresses and _numNfts can be calldata.
	- Initializations with `0` or `false` for newly declared variables consume gas and are unnecessary.
	Examples:
	Line 264
	`uint256 fee = 0;`
	Line 265
	`bool sent = false;`
	Line 199
	`uint256 i = 0;`

Recommendation	Consider defining the above parameters as `calldata` and consider implementing the other recommended fixes.
Resolution	RESOLVED All sections that could be resolved have been resolved.

Issue #08	Inherited init functions not called in the base contract
Severity	INFORMATIONAL
Description	`BaseLaunchpeg` is a contract that follows the Proxy pattern. In this pattern, the constructor is not called; instead, an `init` (initialize) function is defined. The role of this function is to initialize all the variables right after deployment. Within this function also, all the inherited contracts' `init` function must be called. Within `BaseLaunchpeg`, the `__ReentrancyGuard_init()` and `__ERC2981_init()` are not called. This does not have any impact to the logic but it is nevertheless good practice to call all the inherited contracts' `init` functions, in case, at some point during an upgrade, one of these functions would need to be actually called.
Recommendation	Consider calling `__ReentrancyGuard_init()` and `__ERC2981_init()`.
Resolution	RESOLVED

Issue #09	Lack of validation
Severity	INFORMATIONAL
Location	Line 130 `uint256 _maxBatchSize`
Description	Currently, there is no validation for the `_maxBatchSize` to be smaller than the `_collectionSize` as well as a missing validation for the remainder to be zero. (`_collectionSize % maxBatchSize == 0`). The contract also does not support NFTs with a total supply greater than `2**64 - 1`, which as far as we know do not exist. It might make sense to make this requirement explicit in the constructor.
Recommendation	Consider adding validations for the function.
Resolution	ACKNOWLEDGED The client has indicated that the collection size could overflow but this is unlikely to happen. They explained that they also should not restrict the `_maxBatchSize` percentage as people could want odd collection sizes that would give a weird `maxBatchNumber`. People can always buy less NFTs than the `maxBatchSize` so it will not lead to the remainder never being sold.

Issue #10	Typographical error
Severity	INFORMATIONAL
Location	Line 191 `function seedAllowlist`
Description	The contract contains a typographical error.
Recommendation	Consider changing the function name to `seedAllowList`.
Resolution	RESOLVED The client wishes to keep the current spelling.

Issue #11	`withdrawAvax` should have a to parameter in case the owner does not have a fallback function
Severity	INFORMATIONAL
Description	Oftentimes, contracts are owned by a timelock or multisig. Many of such contracts cannot accept AVAX tokens themselves. To allow the NFT contracts to be owned by such contracts, we recommend allowing the owner to withdraw to a separate wallet.
Recommendation	Consider adding a `to` parameter to `withdrawAvax` and sending the owner AVAX there.
Resolution	RESOLVED A `to` parameter has been added.

Issue #12	**`supportsInterface` does not indicate that it supports all interfaces**
Severity	INFORMATIONAL
Description	The `supportsInterface` function presently only indicates that it supports a single interface.
Recommendation	Consider returning the union (using or statements) of all inherited contracts whose supportsInterface return value. Alternatively, OpenZeppelin has an ERC-165 implementation which works with a mapping which you can register supported interfaces in.
Resolution	RESOLVED The client has implemented the recommended union using `or` statements.

2.3 FlatLaunchPeg

The Trader Joe `FlatLaunchPeg` contract implements a fast and efficient NFT token, including a basic launch mechanism. Unlike the `LaunchPeg` contract, the `FlatLaunchPeg` contract contains only two phases:

- `AllowList mint`
- `Public sale`

Users can only buy NFTs at two predetermined prices: the allowlist price and the public sale price. The allowlist price is only accessible by wallets with an explicit allowance to mint several NFTs through the `allowListMint` function. All other users have to pay the public `salePrice` through the `publicSaleMint` function.

2.3.1 Privileges

The following functions can be called by the various privileged roles of the contract:

- `devMint (onlyProjectOwner)`
- `forceReveal (onlyProjectOwner)`
- `initializeJoeFee`
- `initializePhases`
- `setRoyaltyInfo`
- `seedAllowlist`
- `setBaseURI`
- `setUnrevealedURI`
- `setProjectOwner`
- `withdrawAvax`
- `renounceOwnership`
- `transferOwnership`

2.3.2 Issues & Recommendations

Issue #13	**Insufficient check for `maxPerAddressDuringMint` during `publicSaleMint`**
Severity	HIGH SEVERITY
Description	Within the function `publicSaleMint`, there is a check for the case `if (_quantity > maxPerAddressDuringMint)`. However, this can be gamed because there is no check for `numberMinted[msg.sender]`: `if (numberMinted(msg.sender) + _quantity > maxPerAddressDuringMint)` is missing. Users should note that malicious parties can still bypass the check through simple Sybil attacks, even with a recommended per-address accumulation check.
Recommendation	Consider adding a validation if this is not intentional. The client can also resolve this issue on the note that the present check is intentional, and they intend for this check to be per transaction.
Resolution	RESOLVED `numberMinted` is now used.

Issue #14	Irregularities and unexpected behaviour during `Mint` events
Severity	INFORMATIONAL
Description	During the emission of the `Mint` event, `allowListMint` uses the `mintListPrice` as a parameter. However, `publicSaleMint` is using `total` as a parameter. This difference is inconsistent and could confuse off-chain aggregators. Secondly, the `Mint` event does not adhere to checks-effects-interactions, causing the last parameter to potentially emit an erroneous variable if the user reenters through `_refundIfOver`. In this case, the last event variable would be too high for the first mint.
Recommendation	Consider following a clear structure and replacing `total` with `salePrice`. Consider also re-ordering the event emission to be above the `_refundIfOver` call in the `mint` functions.
Resolution	RESOLVED

Issue #15	Lack of validation
Severity	INFORMATIONAL

Description	Within the initialize function there is a lack of validation for the following parameters:
	
	`uint256 _maxBatchSize`
	`uint256 _batchRevealSize`
	
	The batch sizes should be smaller or equal to _collectionSize.
	
	`uint256 _amountForDevs`
	
	Should be smaller than _collectionSize and kept in a reasonable range.
	
	`uint256 _salePrice`
	`uint256 _mintlistPrice`
	
	Consider if it is desired that `_mintlistPrice` is always cheaper than `_salePrice`, and if yes, validate this requirement: `_salePrice > _mintlistPrice`.

Recommendation	Consider validating these parameters.
Resolution	RESOLVED All validation has been included (some in the `BaseLaunchPeg` contract).

Issue #16	Duplicate import
Severity	INFORMATIONAL
Description	The `import "./LaunchpegErrors.sol"` is unnecessary since the `BaseLaunchPeg` contract already imports it.
Recommendation	Consider removing the duplicate import.
Resolution	RESOLVED

Issue #17	Gas optimizations
Severity	INFORMATIONAL
Description	Within the `initializeBaseLaunchpeg` function, the `_name` and `_symbol` parameters can be declared as `calldata` to save gas.
Recommendation	Consider defining these parameters as `calldata`.
Resolution	RESOLVED The client has indicated they prefer to keep their `init` function `memory`.

2.4 LaunchPeg

The Trader Joe LaunchPeg contract implements a fast and efficient NFT launch mechanism. Each NFT sale organised through a LaunchPeg contract has three phases:

Auction (a Dutch Auction that decreases the price every auctionDropInterval by auctionDropPerStep)

AllowList phase: Allows only allowlisted addresses to purchase NFTs up to their allowance at a specific allowlist discount

Public sale: Allows anyone to purchase and mint the remaining NFTs up to their allowance at a specific public sale discount

2.4.1 Privileges

The following functions can be called by the various privileged roles of the contract:

- `devMint (onlyProjectOwner)`
- `forceReveal (onlyProjectOwner)`
- `initializeJoeFee`
- `setRoyaltyInfo`
- `seedAllowlist`
- `setBaseURI`
- `setUnrevealedURI`
- `setProjectOwner`
- `withdrawAvax`
- `renounceOwnership`
- `transferOwnership`

2.4.2 Issues & Recommendations

Issue #18	Missing safeguards on `initializePhases`
Severity	LOW SEVERITY
Description	The `initializePhases` is used to initialise the phases in which minting is happening as the following order of minting phases must be kept: Auction -> Mintlist -> Public. Within this function, there are checks for the order to be kept but there is no check for starting the order from the past, meaning that some of the phases can be skipped by setting them in the past. Additionally, the `revealStartTime` and `revealInterval` have also missing safeguards which can lead to an undesired outcome. These `revealStartTime` can be set in the past or in the far future, making the contracts unusable. The Paladin team has seen in the past misplaced 0s that led to a wrong configuration. The `revealInterval` as well can be set very high, making a batch reveal very slow.
Recommendation	Consider adding a check that `auctionSaleStartTime > block.timestamp`. Additionally, consider adding some safeguards checks for `revealStartTime` and `revealInterval` to not have abnormal values.
Resolution	RESOLVED The client has indicated that being able to set the `batchReveal` variables to be in the past is intended. `auctionSaleStartTime` validation has been added.

Issue #19	Lack of validation
Severity	INFORMATIONAL

Description	Throughout the contract, there is a lack of validation on various functions. To keep the report size reasonable, all missing validations are consolidated within this issue.
	
	Line 200
	`uint256 _auctionDropInterval`
	
	The `_auctionDropInterval` does not have any validation. If it is unnecessarily high, steps could round down to zero and therefore there would not be any discount during the auction phase.
	
	Line 224
	`_mintlistDiscountPercent > 10000 ||`
	`_publicSaleDiscountPercent > 10000`
	
	Currently, there is no check that `_mintListDiscountPercent > _publicSaleDiscountPercent`. If this is not intended, the `publicSaleMint` price could end up being cheaper than the `allowListMint` price.
	
	These last statements amongst various other statements could also use the `BASIS_POINT_PRECISION` constant instead of a hardcoded `10 000`.

Recommendation	Consider adding extra checks to the above variables.
Resolution	RESOLVED The client has indicated that a check for `_mintListDiscountPercent > _publicSaleDiscountPercent` is also not desired.

Issue #20	Duplicate imports
Severity	INFORMATIONAL

Description	The contract imports various contracts which are already imported within BaseLaunchPeg.
	
	`import "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol"`
	
	`import "erc721a-upgradeable/contracts/ERC721AUpgradeable.sol"`
	
	`import "./BatchReveal.sol"`
	
	`import "./LaunchpegErrors.sol"`
	
	These duplicated imports are not necessary.

Recommendation	Consider removing the duplicated imports to keep the contract short and simple.
Resolution	RESOLVED

Issue #21	Unexpected behaviour during `Mint` events in case of reentrancy
Severity	INFORMATIONAL
Description	The `Mint` event is not written in checks-effects-interactions causing the last parameter to potentially emit a wrongful variable if the user reenters through `_refundIfOver`. In this case, the last event variable would be too high for the first mint.
Recommendation	Consider re-ordering the event emission to above the `_refundIfOver` call in the `mint` functions.
Resolution	RESOLVED The function calls have been re-ordered.

2.5 BatchReveal

`BatchReveal` is an abstract contract that implements a gas efficient way of revealing NFT URIs gradually. It was developed by a known NFT collection called Tubby Cats. The contract uses a Fisher–Yates algorithm, an optimal algorithm for shuffling a list of items that gives a certain token url for an id, and the token url is generated based on a batched-reveal approach. After every `revealBatchSize`, anyone can call `revealNextBatch` method and the minted batch is revealed.

The batch reveal approach is achieved using a randomness or pseudo randomness strategy. Essentially, there are two types of implementations: using Chainlink VRF or using on-chain pseudo-randomness by blockhash. The Trader Joe team has chosen to use the on-chain pseudo-randomness as VRF is not yet present on AVAX network.

The on-chain pseudo-randomness is very risky for use in NFT reveals as the outcome can be predicted by the miners/exploiters to get the best rewards.

The Paladin team did not discover a very obvious way to manipulate the pseudo-randomness as the Trader Joe team has used variables like `tx.gasprice`, `block.difficulty`, etc. in their randomness — parameters that are very expensive to be manipulated by the validators. This, alongside with the batch reveal and a restriction for the reveal to be called only by EOAs, assures an almost fair reveal for the users.

2.5.1 Issues & Recommendations

Issue #22	`_getShuffledTokenId` can run out of gas
Severity	MEDIUM SEVERITY
Description	The `_getShuffledTokenId` UI function executes a complex algorithm whenever the frontend wants to figure out the NFT URI. If there are too many batches, this shuffling algorithm would certainly run out of gas.
Recommendation	Consider limiting the number of batches to a number that has proven to work on Avalanche.
Resolution	RESOLVED The client has indicated they will carefully test this on Avalanche to figure out a reasonable reveal limit.

Issue #23	Last incomplete batch cannot be revealed by users
Severity	LOW SEVERITY
Description	Every NFT that inherits the `BatchReveal` approach for revealing the metadata, uses a batch of number `revealBatchSize` to reveal the already minted NFTs. The mechanism works in the following way: Users mint `revealBatchSize` number of tokens, someone calls the reveal method, and after that, the `tokenURI` function returns a shuffled token url for the metadata. The problem with this approach is if users stop minting while the batch size has not been reached.
Recommendation	Consider adding a user function where if the mint period has expired, they can reveal any unrevealed batch.
Resolution	PARTIALLY RESOLVED The client has indicated that the privileged `forceReveal` is sufficient for their purposes. This issue has been marked as partially resolved as the issue pinpoints the fact that users cannot trigger this.

Issue #24	Tokenomics: Premature revealing could incentivize waiting behaviour
Severity	LOW SEVERITY
Description	The NFTs that implement `BatchReveal` allow for a batch to be revealed as soon as it is minted. This means that the mint is still ongoing while NFTs are already being revealed. This incentivizes waiting behaviour, as we demonstrate in the following tokenomical proof of concept: Let's say the mint cost is $10 on a collection of 6 NFTs that is revealed in two batches (of three NFTS). Three of these NFTs, the "common" ones, have an intrinsic value to the minter of $9. The other three "rare" ones have an intrinsic value to the minter of $11. In other words, in a random assignment of NFTs, the expected value for the minter would be $10, eg. they would be indifferent about minting. However, after the initial batch has been revealed, this indifference can change. In case the first batch contains the 3 common NFTs, the expected value for minting suddenly becomes $1, as the minter will always get a rare NFT. As the payoff for minting early is zero and users are not forced to mint in the second batch, all users are incentivized to wait until the second batch to execute their mints (assuming a budgetary constraint). Such behaviour might not be desired.
Recommendation	Consider whether this poses a threat to the reveal mechanism. If so, a single reveal might need to be considered or large enough batch sizes to make the statistical advantage of waiting negligible.
Resolution	ACKNOWLEDGED The client has indicated that they will generally have large enough batch sizes to minimize the impact of this.

2.6 ERC721A

`ERC721A` is a popular ERC721 NFT implementation. Its main purpose is to be able to mint a large amount of NFTs without the gas cost increasing with the number of NFTs bought. One thing to note about ERC721A is that it enables only minting sequential ids for the `tokenIDs`.

Paladin audited the following version of ERC721A: https://github.com/chiru-labs/ERC721A/blob/b17a1c4310d24ea75700211fdd50900f8a26d344/contracts/ERC721A.sol

The version from the TraderJoe is the upgradable version of what Paladin team has audited with small modifications of mint and safeMint combined into 1 function with a parameter safe that triggers the safeMint functionality.

2.6.1 Issues & Recommendations

Although small informational findings were found while auditing the ERC721A dependency, they have been omitted from the report as resolving them would provide no value to the JoePeg system and this is considered an external dependency which recommend leaving unchanged.

2.7 LaunchPegErrors

`LaunchPegErrors` file is a simple collection of solidity errors used throughout the codebase. Within recent Solidity versions, using such a collection has become best practice compared to revert statements that return an error string. The reason for this is because errors are significantly more gas efficient to revert with.

2.7.1 Issues & Recommendations

Issue #25	Unused functionality
Severity	INFORMATIONAL

Location	Line 10
	`error Launchpeg__InvalidAuctionSaleDuration();`
	
	Line 25
	`error Launchpeg__TooManyAlreadyMintedBeforeDevMint();`

Description	The above functions are not used anywhere within the contract and increase the size of the contract unnecessarily.
Recommendation	Consider removing the functions to keep the contract short and simple.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY