Smart Contract

Security Assessment

For GoldRush

Degens

01 Apr 2023

Table of Contents

DISCLAIMER

This independent audit has been conducted 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 the auditor.

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 the auditor is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will the auditor 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.

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

Executive Summary

Severity	Found
High	0
Medium	2
Low	15
Informational	57
Total	74

We performed an independent technical audit to identify Smart Contracts uncertainties. This shall protect the code from illegitimate authorization attempts or external & internal threats of any type. This also ensures end-to-end proofing of the contract from frauds. The audit was performed semi-manually. We analyzed the Smart Contracts code line-by-line and used an automation tool to report any suspicious code.

The following tools were used:

- Truffle
- Remix IDE
- Slither

Overview

This report has been prepared for GoldRush Degens. This audit provides a user-centered examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

Summary

Project Name	Gold Rush Degens
Platform	Ethereum
Language	Solidity

Contracts Assessed

Name	Location
GoldRushDegens.sol	Not Published
Strings.sol	In GoldRushDegens contract
Math.sol	In GoldRushDegens contract
SafeMath.sol	In GoldRushDegens contract
IERC20.sol	In GoldRushDegens contract
ReentrancyGuardUpgradeable.sol	In GoldRushDegens contract

Name	Location
Address.sol	In GoldRushDegens contract
IERC721Upgradeable.sol	In GoldRushDegens contract
IERC721EnumerableUpgradeable.sol	In GoldRushDegens contract
IERC721ReceiverUpgradeable.sol	In GoldRushDegens contract
IERC721MetadataUpgradeable.sol	In GoldRushDegens contract
ERC721Upgradeable.sol	In GoldRushDegens contract
ERC721EnumerableUpgradeable.sol	In GoldRushDegens contract
IERC165Upgradeable.sol	In GoldRushDegens contract
ERC165Upgradeable.sol	In GoldRushDegens contract
IERC20Permit.sol	In GoldRushDegens contract
MerkleProof.sol	In GoldRushDegens contract
MathUpgradeable.sol	In GoldRushDegens contract
StringsUpgradeable.sol	In GoldRushDegens contract
AddressUpgradeable.sol	In GoldRushDegens contract
Initializable.sol	In GoldRushDegens contract
ContextUpgradeable.sol	In GoldRushDegens contract
OwnableUpgradeable.sol	In GoldRushDegens contract
SafeERC20.sol	In GoldRushDegens contract

Findings Summary

Severity	Found
High	0
Medium	2
Low	15
Informational	57
Total	74

Classification of Issues

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.

Manual Review

Issues Checking Status

Issue Description	Checking Status
Compiler errors	PASS
Race conditions and Reentrancy. Cross-function race conditions.	PASS
Possible delays in data delivery.	PASS
Oracle calls.	PASS
Front running.	PASS
Timestamp dependence.	PASS
Integer Overflow and Underflow.	PASS
DoS with Revert.	PASS
DoS with block gas limit.	PASS
Methods execution permissions.	PASS
Economy model of the contract.	PASS
The impact of the exchange rate on the logic.	PASS
Private user data leaks.	PASS
Malicious Event log.	PASS
Scoping and Declarations.	PASS
Uninitialized storage pointers.	PASS

Arithmetic accuracy.	PASS
Design Logic.	PASS
Cross-function race conditions.	PASS
Safe Open Zeppelin contracts implementation and usage.	PASS
Fallback function security.	PASS

Audit Findings

Severity	Medium
Contract	GoldRushDegens.sol
Description	Claim_End_Timestamp goes to April 3 not April 30
Code Snippet	uint256 public constant CLAIM_END_TIMESTAMP = 1680489599; // April 30th, 2023 11:59:59 PM UTC
Recommendation	The variable Claim_End_Timestamp is set at April 3rd, however, the comment indicates that the end date should be April 30th. If this was simply a typo, or plans changed, you can safely disregard this finding. The variable is also marked constant and therefore cannot be changed after deployment without requiring an upgrade. Consider unmarking it constant and including a set function.
Status	RESOLVED

Audit Findings

Severity	**Low**
Contract	GoldRushDegens.sol
Description	_burn function fails to transfer to zero address

Code Snippet	function _burn(uint256 tokenId)
	internal override {
	super._burn(tokenId);
	
	if (bytes(_tokenURIs[tokenId]).length
	!= 0) {
	delete _tokenURIs[tokenId];
	}
	}

Recommendation	_burn is an internal function that calls _burn from ERC721Upgradeable. Because it is internal, no calls can be made to it from outside the contract, unless the contract calling it is derived from it. Normally, in order to call this function, there must be a function that uses it within GoldRushDegens. Furthermore, if the caller attempts to transfer the token to the zero address using transfer or safeTransfer functions, these functions require the 'to' address not to be address(0), so the call would fail. Because the utility of this function is unclear, we recommend reviewing it, modifying it, or removing depending on the needs of the project.
Status	RESOLVED

Audit Findings

Severity	Informational
Contract	GoldRushDegens.sol
Description	Weak PRNG (Use of blockhash)
Code Snippet	uint256 randomNumber = uint256(keccak256(abi.encode(to, tx.gasprice, block.number, block.timestamp, block.prevrandao, blockhash(block.number - 1), address(this), updatedNumAvailableTokens))) % MAX_NFTS;
Recommendation	Generally speaking, it is advised not to use block.timestamp, blockhash, etc. to generate pseudorandom numbers as this can be exploited by miners. It is acknowledged that isEOA mitigates the risk to a certain degree.
Status	ACKNOWLEDGED

Audit Findings

Severity	Informational
Contract	GoldRushDegens.sol
Description	No set functions for important variables
Code Snippet	IERC20 private _stablecoin
Recommendation	It is acknowledged that because the contract is upgradeable, changes can be made to the contract by redeploying through the proxy contract. However, in the interest of cost efficiency, it is still more expensive to do so than to simply include a set function.
Status	RESOLVED

Audit Findings

Severity	Informational
Contract	GoldRushDegens.sol
Description	No fallback function for receiving native token.
Code Snippet	N/A
Recommendation	It is clear that the mint contract is designed to route a stablecoin to the receiver contract instead of ETH, however, by not including a fallback function (which is just good practice), the contract will not know what to do with stuck ETH in the event ETH gets sent to the contract. Consider including a fallback receiver function and possibly a withdraw function specifically for withdrawing stuck native tokens.
Status	RESOLVED

Functional Test Status

Function Name	Type/Return Type	Score
Math/MathUpgradeable		
average	internal	PASS
ceilDiv	internal	PASS
log10	internal	PASS
log2	internal	PASS
log256	internal	PASS
max	internal	PASS
min	internal	PASS
mulDiv	internal	PASS
sqrt	internal	PASS
IERC20		
allowance	read/external	PASS
approve	write/external	PASS
balanceOf	read/external	PASS
totalSupply	read/external	PASS
transfer	write/external	PASS
transferFrom	write/external	PASS
Context/ContextUpgradeable		

Function Name	Type/Return Type	Score
_msgData	internal	PASS
_msgSender	internal	PASS
OwnableUpgradeable		
_checkOwner	internal	PASS
transferOwnership	write/public	PASS
owner	read/public	PASS
renounceOwnership	write/public	PASS
Address/AddressUpgradeable		
functionCall	internal	PASS
functionCallWithValue	internal	PASS
functionDelegateCall	internal	PASS
functionStaticCall	internal	PASS
isContract	internal	PASS
sendValue	internal	PASS
verifyCallResult	internal	PASS
ERC721MetadataUpgradeable		
name	read/external	PASS
symbol	read/external	PASS
tokenURI	read/external	PASS

IERC721ReceiverUpgradeable		
onERC721Received	external	PASS
IERC721Upgradeable		
approve	write/external	PASS
balanceOf	read/external	PASS
getApproved	read/external	PASS
isApprovedForAll	read/external	PASS
ownerOf	read/external	PASS
safeTransferFrom	write/external	PASS
setApprovalForAll	write/external	PASS
transferFrom	write/external	PASS
ERC721Upgradeable		
_afterTokenTransfer	internal	PASS
_approve	internal	PASS
_baseURI	internal	PASS
_beforeTokenTransfer	internal	PASS
_burn	internal	PASS
_checkOnERC721Received	private	``PASS
_exists	internal	PASS
_isApprovedOrOwner	internal	PASS
_mint	internal	PASS
_ownerOf	internal	PASS
_requireMinted	internal	PASS

MerkleProof		
_efficientHash	private	PASS
_hashPair	private	PASS
multiProofVerify	internal	PASS
multiProofVerifyCalldata	internal	PASS
processMultiProof	internal	PASS
processMultiProofCalldata	internal	PASS
verify	internal	PASS
verifyCalldata	internal	PASS
Strings/StringsUpgradeable		
toHexString	internal	PASS
toString	internal	PASS
IERC721Enumerable		
tokenByIndex	external	PASS
tokenOfOwnerByIndex	external	PASS
totalSupply	external	PASS
ERC721EnumerableUpgradeable		
_init	internal	PASS
_init_unchained	internal	PASS
_addTokenToAllTokensEnumeration	private	PASS
addTokenToOwnerEnumeration	private	PASS

beforeTokenTransfer	internal	PASS
removeTokenFromAllTokensEnumeration	private	PASS
removeTokenFromOwnerEnumeration	private	PASS
processMultiProof	internal	PASS
processMultiProofCalldata	internal	PASS
verify	internal	PASS
verifyCalldata	internal	PASS
Strings/StringsUpgradeable		
toHexString	internal	PASS
toString	internal	PASS
IERC721Enumerable		
tokenByIndex	external	PASS
tokenOfOwnerByIndex	external	PASS
totalSupply	external	PASS
IERC721MetadataUpgradeable		
name	external	PASS
symbol	external	PASS
tokenURI	external	PASS
ERC721EnumerableUpgradeable		
_init	internal	PASS
_init_unchained	internal	PASS

IERC20Permit		
nonces	external	PASS
permit	external	PASS
SafeERC20		
callOptionalReturn	private	PASS
safeApprove	internal	PASS
safeDecreaseAllowance	internal	PASS
safeIncreaseAllowance	internal	PASS
safePermit	internal	PASS
safeTransfer	internal	PASS
safeTransferFrom	internal	PASS
Initializable		
_disableInitializers	internal	PASS
_getInitializedVersion	internal	PASS
_isInitializing	internal	PASS
IERC165Upgradeable		
supportsInterface	external	PASS
ERC165Upgradeable		
_init	internal	PASS
_init_unchained	internal	PASS
_mint	internal	PASS
_ownerOf	internal	PASS
_requireMinted	internal	PASS

_safeMint	internal	PASS
_safeTransfer	internal	PASS
_setApprovalForAll	internal	PASS
_transfer	internal	PASS
approve	public	PASS
balanceOf	public	PASS
constructor	public	PASS
getApproved	public	PASS
isApprovedForAll	public	PASS
name	public	PASS
ownerOf	public	PASS
safeTransferFrom	public	PASS
setApprovalForAll	public	PASS
supportsInterface	public	PASS
symbol	public	PASS
tokenURI	public	PASS
transferFrom	public	PASS
GoldRushDegens		
_baseURI	internal	PASS
_burn	write/internal	PASS
_constructTokenURI	write/internal	PASS
_getRandomTokenId	private	PASS
_mintRandomNFT	internal	PASS

_setTokenURI	write/internal	PASS
activateDropPhase	write/external	PASS
claim	write/external	PASS
getCurrentDropPhase	write/external	PASS
getDropPhaseById	write/external	PASS
initialize	public	PASS
isEOA	internal	PASS
mint	write/external	PASS
mintUnclaimedReservedNFTs	write/external	PASS
setDropPhase	write/external	PASS
tokenURI	public	PASS
updateWhitelist	write/external	PASS

Omitted Results

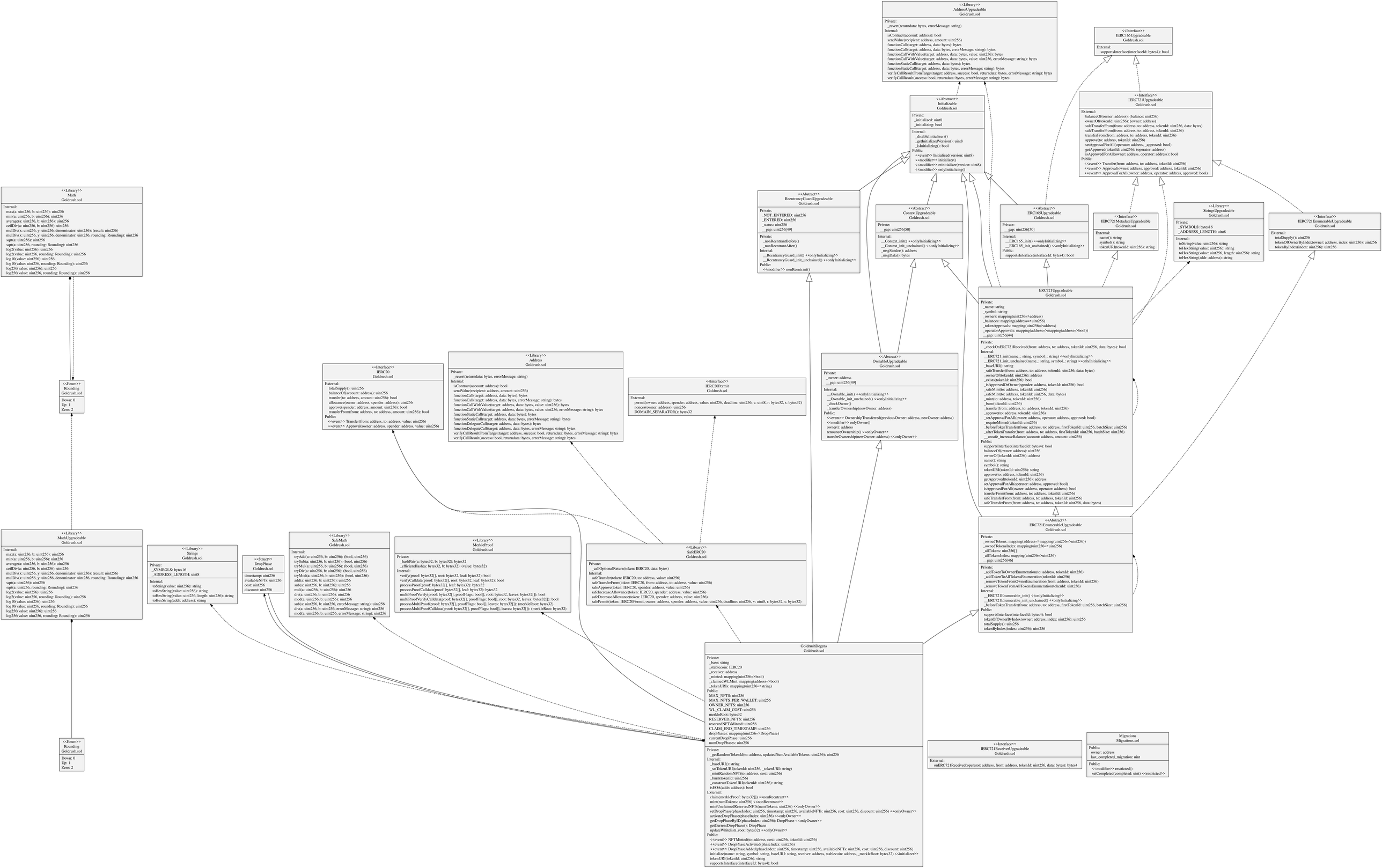
Note: Any issues that have been omitted from this report have been deemed by the reviewing team as irrelevant, inapplicable, and/or negligible to the proper functioning of this contract. Thus, any omitted issues can be safely ignored.

Automated Review

Solidity Static Analysis

Issue	Severity
Gas requirement of function GoldrushDegens.initialize is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)Pos: 79:4:	Informational
Gas requirement of function GoldrushDegens.claim is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)Pos: 130:4:	Informational
Gas requirement of function GoldrushDegens.mint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)Pos: 169:4:	Informational
Gas requirement of function GoldrushDegens.mintUnclaimedReservedNFTs is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)Pos: 205:4:	Informational

Issue	Severity
Block hash:Use of "blockhash": "blockhash(uint blockNumber)" is used to access the last 256 block hashes. A miner computes the block hash by "summing up" the information in the current block mined. By "summing up" the information cleverly, a miner can try to influence the outcome of a transaction in the current block. This is especially easy if there are only a small number of equally likely outcomes.Pos: 280:20:	Informational
Block timestamp:Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.	Informational


<<Library>>
Math
Goldrush.sol
Internal:
max(a: uint256, b: uint256): uint256
min(a: uint256, b: uint256): uint256
average(a: uint256, b: uint256): uint256
ceilDiv(a: uint256, b: uint256): uint256
mulDiv(x: uint256, y: uint256, denominator: uint256): (result: uint256)
mulDiv(x: uint256, y: uint256, denominator: uint256, rounding: Rounding): uint256
sqrt(a: uint256): uint256
sqrt(a: uint256, rounding: Rounding): uint256
log2(value: uint256): uint256
log2(value: uint256, rounding: Rounding): uint256
log10(value: uint256): uint256
log10(value: uint256, rounding: Rounding): uint256
log256(value: uint256): uint256
log256(value: uint256, rounding: Rounding): uint256
<<Enum>>
Rounding
Goldrush.sol
Down: 0
Up: 1
Zero: 2
<<Library>>
MathUpgradeable
Goldrush.sol
Internal:
max(a: uint256, b: uint256): uint256
min(a: uint256, b: uint256): uint256
average(a: uint256, b: uint256): uint256
ceilDiv(a: uint256, b: uint256): uint256
mulDiv(x: uint256, y: uint256, denominator: uint256): (result: uint256)
mulDiv(x: uint256, y: uint256, denominator: uint256, rounding: Rounding): uint256
sqrt(a: uint256): uint256
sqrt(a: uint256, rounding: Rounding): uint256
log2(value: uint256): uint256
log2(value: uint256, rounding: Rounding): uint256
log10(value: uint256): uint256
log10(value: uint256, rounding: Rounding): uint256
log256(value: uint256): uint256
log256(value: uint256, rounding: Rounding): uint256
<<Enum>>
Rounding
Goldrush.sol
Down: 0
Up: 1
Zero: 2
<<Interface>>
IERC20
Goldrush.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Library>>
Address
Goldrush.sol
Private:
_revert(returndata: bytes, errorMessage: string)
Internal:
isContract(account: address): bool
sendValue(recipient: address, amount: uint256)
functionCall(target: address, data: bytes): bytes
functionCall(target: address, data: bytes, errorMessage: string): bytes
functionCallWithValue(target: address, data: bytes, value: uint256): bytes
functionCallWithValue(target: address, data: bytes, value: uint256, errorMessage: string): bytes
functionStaticCall(target: address, data: bytes): bytes
functionStaticCall(target: address, data: bytes, errorMessage: string): bytes
functionDelegateCall(target: address, data: bytes): bytes
functionDelegateCall(target: address, data: bytes, errorMessage: string): bytes
verifyCallResultFromTarget(target: address, success: bool, returndata: bytes, errorMessage: string): bytes
verifyCallResult(success: bool, returndata: bytes, errorMessage: string): bytes
<<Interface>>
IERC20Permit
Goldrush.sol
External:
permit(owner: address, spender: address, value: uint256, deadline: uint256, v: uint8, r: bytes32, s: bytes32)
nonces(owner: address): uint256
DOMAIN_SEPARATOR(): bytes32
<<Library>>
StringsUpgradeable
Goldrush.sol
Private:
_SYMBOLS: bytes16
_ADDRESS_LENGTH: uint8
Internal:
toString(value: uint256): string
toHexString(value: uint256): string
toHexString(value: uint256, length: uint256): string
toHexString(addr: address): string
<<Library>>
Strings
Goldrush.sol
Private:
_SYMBOLS: bytes16
_ADDRESS_LENGTH: uint8
Internal:
toString(value: uint256): string
toHexString(value: uint256): string
toHexString(value: uint256, length: uint256): string
toHexString(addr: address): string
<<Struct>>
DropPhase
Goldrush.sol
timestamp: uint256
availableNFTs: uint256
cost: uint256
discount: uint256
<<Library>>
SafeMath
Goldrush.sol
Internal:
tryAdd(a: uint256, b: uint256): (bool, uint256)
trySub(a: uint256, b: uint256): (bool, uint256)
tryMul(a: uint256, b: uint256): (bool, uint256)
tryDiv(a: uint256, b: uint256): (bool, uint256)
tryMod(a: uint256, b: uint256): (bool, uint256)
add(a: uint256, b: uint256): uint256
sub(a: uint256, b: uint256): uint256
mul(a: uint256, b: uint256): uint256
div(a: uint256, b: uint256): uint256
mod(a: uint256, b: uint256): uint256
sub(a: uint256, b: uint256, errorMessage: string): uint256
div(a: uint256, b: uint256, errorMessage: string): uint256
mod(a: uint256, b: uint256, errorMessage: string): uint256
<<Library>>
MerkleProof
Goldrush.sol
Private:
_hashPair(a: bytes32, b: bytes32): bytes32
_efficientHash(a: bytes32, b: bytes32): (value: bytes32)
Internal:
verify(proof: bytes32[], root: bytes32, leaf: bytes32): bool
verifyCalldata(proof: bytes32[], root: bytes32, leaf: bytes32): bool
processProof(proof: bytes32[], leaf: bytes32): bytes32
processProofCalldata(proof: bytes32[], leaf: bytes32): bytes32
multiProofVerify(proof: bytes32[], proofFlags: bool[], root: bytes32, leaves: bytes32[]): bool
multiProofVerifyCalldata(proof: bytes32[], proofFlags: bool[], root: bytes32, leaves: bytes32[]): bool
processMultiProof(proof: bytes32[], proofFlags: bool[], leaves: bytes32[]): (merkleRoot: bytes32)
processMultiProofCalldata(proof: bytes32[], proofFlags: bool[], leaves: bytes32[]): (merkleRoot: bytes32)
<<Library>>
SafeERC20
Goldrush.sol
Private:
_callOptionalReturn(token: IERC20, data: bytes)
Internal:
safeTransfer(token: IERC20, to: address, value: uint256)
safeTransferFrom(token: IERC20, from: address, to: address, value: uint256)
safeApprove(token: IERC20, spender: address, value: uint256)
safeIncreaseAllowance(token: IERC20, spender: address, value: uint256)
safeDecreaseAllowance(token: IERC20, spender: address, value: uint256)
safePermit(token: IERC20Permit, owner: address, spender: address, value: uint256, deadline: uint256, v: uint8, r: bytes32, s: bytes32)
<<Library>>
AddressUpgradeable
Goldrush.sol
Private:
_revert(returndata: bytes, errorMessage: string)
Internal:
isContract(account: address): bool
sendValue(recipient: address, amount: uint256)
functionCall(target: address, data: bytes): bytes
functionCall(target: address, data: bytes, errorMessage: string): bytes
functionCallWithValue(target: address, data: bytes, value: uint256): bytes
functionCallWithValue(target: address, data: bytes, value: uint256, errorMessage: string): bytes
functionStaticCall(target: address, data: bytes): bytes
functionStaticCall(target: address, data: bytes, errorMessage: string): bytes
verifyCallResultFromTarget(target: address, success: bool, returndata: bytes, errorMessage: string): bytes
verifyCallResult(success: bool, returndata: bytes, errorMessage: string): bytes
<<Interface>>
IERC165Upgradeable
Goldrush.sol
External:
supportsInterface(interfaceId: bytes4): bool
<<Abstract>>
Initializable
Goldrush.sol
Private:
_initialized: uint8
_initializing: bool
Internal:
_disableInitializers()
_getInitializedVersion(): uint8
_isInitializing(): bool
Public:
<<event>> Initialized(version: uint8)
<<modifier>> initializer()
<<modifier>> reinitializer(version: uint8)
<<modifier>> onlyInitializing()
<<Interface>>
IERC721Upgradeable
Goldrush.sol
External:
balanceOf(owner: address): (balance: uint256)
ownerOf(tokenId: uint256): (owner: address)
safeTransferFrom(from: address, to: address, tokenId: uint256, data: bytes)
safeTransferFrom(from: address, to: address, tokenId: uint256)
transferFrom(from: address, to: address, tokenId: uint256)
approve(to: address, tokenId: uint256)
setApprovalForAll(operator: address, _approved: bool)
getApproved(tokenId: uint256): (operator: address)
isApprovedForAll(owner: address, operator: address): bool
Public:
<<event>> Transfer(from: address, to: address, tokenId: uint256)
<<event>> Approval(owner: address, approved: address, tokenId: uint256)
<<event>> ApprovalForAll(owner: address, operator: address, approved: bool)
<<Abstract>>
ReentrancyGuardUpgradeable
Goldrush.sol
Private:
_NOT_ENTERED: uint256
_ENTERED: uint256
_status: uint256
__gap: uint256[49]
Private:
_nonReentrantBefore()
_nonReentrantAfter()
Internal:
__ReentrancyGuard_init() <<onlyInitializing>>
__ReentrancyGuard_init_unchained() <<onlyInitializing>>
Public:
<<modifier>> nonReentrant()
<<Abstract>>
ContextUpgradeable
Goldrush.sol
Private:
__gap: uint256[50]
Internal:
__Context_init() <<onlyInitializing>>
__Context_init_unchained() <<onlyInitializing>>
_msgSender(): address
_msgData(): bytes
<<Abstract>>
ERC165Upgradeable
Goldrush.sol
Private:
__gap: uint256[50]
Internal:
__ERC165_init() <<onlyInitializing>>
__ERC165_init_unchained() <<onlyInitializing>>
Public:
supportsInterface(interfaceId: bytes4): bool
<<Interface>>
IERC721MetadataUpgradeable
Goldrush.sol
External:
name(): string
symbol(): string
tokenURI(tokenId: uint256): string
<<Interface>>
IERC721EnumerableUpgradeable
Goldrush.sol
External:
totalSupply(): uint256
tokenOfOwnerByIndex(owner: address, index: uint256): uint256
tokenByIndex(index: uint256): uint256
<<Abstract>>
OwnableUpgradeable
Goldrush.sol
Private:
_owner: address
__gap: uint256[49]
Internal:
__Ownable_init() <<onlyInitializing>>
__Ownable_init_unchained() <<onlyInitializing>>
_checkOwner()
_transferOwnership(newOwner: address)
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
owner(): address
renounceOwnership() <<onlyOwner>>
transferOwnership(newOwner: address) <<onlyOwner>>
ERC721Upgradeable
Goldrush.sol
Private:
_name: string
_symbol: string
_owners: mapping(uint256=>address)
_balances: mapping(address=>uint256)
_tokenApprovals: mapping(uint256=>address)
_operatorApprovals: mapping(address=>mapping(address=>bool))
__gap: uint256[44]
Private:
_checkOnERC721Received(from: address, to: address, tokenId: uint256, data: bytes): bool
Internal:
__ERC721_init(name_: string, symbol_: string) <<onlyInitializing>>
__ERC721_init_unchained(name_: string, symbol_: string) <<onlyInitializing>>
_baseURI(): string
_safeTransfer(from: address, to: address, tokenId: uint256, data: bytes)
_ownerOf(tokenId: uint256): address
_exists(tokenId: uint256): bool
_isApprovedOrOwner(spender: address, tokenId: uint256): bool
_safeMint(to: address, tokenId: uint256)
_safeMint(to: address, tokenId: uint256, data: bytes)
_mint(to: address, tokenId: uint256)
_burn(tokenId: uint256)
_transfer(from: address, to: address, tokenId: uint256)
_approve(to: address, tokenId: uint256)
_setApprovalForAll(owner: address, operator: address, approved: bool)
_requireMinted(tokenId: uint256)
_beforeTokenTransfer(from: address, to: address, firstTokenId: uint256, batchSize: uint256)
_afterTokenTransfer(from: address, to: address, firstTokenId: uint256, batchSize: uint256)
__unsafe_increaseBalance(account: address, amount: uint256)
Public:
supportsInterface(interfaceId: bytes4): bool
balanceOf(owner: address): uint256
ownerOf(tokenId: uint256): address
name(): string
symbol(): string
tokenURI(tokenId: uint256): string
approve(to: address, tokenId: uint256)
getApproved(tokenId: uint256): address
setApprovalForAll(operator: address, approved: bool)
isApprovedForAll(owner: address, operator: address): bool
transferFrom(from: address, to: address, tokenId: uint256)
safeTransferFrom(from: address, to: address, tokenId: uint256)
safeTransferFrom(from: address, to: address, tokenId: uint256, data: bytes)
<<Abstract>>
ERC721EnumerableUpgradeable
Goldrush.sol
Private:
_ownedTokens: mapping(address=>mapping(uint256=>uint256))
_ownedTokensIndex: mapping(uint256=>uint256)
_allTokens: uint256[]
_allTokensIndex: mapping(uint256=>uint256)
__gap: uint256[46]
Private:
_addTokenToOwnerEnumeration(to: address, tokenId: uint256)
_addTokenToAllTokensEnumeration(tokenId: uint256)
_removeTokenFromOwnerEnumeration(from: address, tokenId: uint256)
_removeTokenFromAllTokensEnumeration(tokenId: uint256)
Internal:
__ERC721Enumerable_init() <<onlyInitializing>>
__ERC721Enumerable_init_unchained() <<onlyInitializing>>
_beforeTokenTransfer(from: address, to: address, firstTokenId: uint256, batchSize: uint256)
Public:
supportsInterface(interfaceId: bytes4): bool
tokenOfOwnerByIndex(owner: address, index: uint256): uint256
totalSupply(): uint256
tokenByIndex(index: uint256): uint256
GoldrushDegens
Goldrush.sol
Private:
_base: string
_stablecoin: IERC20
_receiver: address
_minted: mapping(uint256=>bool)
_claimedWLMint: mapping(address=>bool)
_tokenURIs: mapping(uint256=>string)
Public:
MAX_NFTS: uint256
MAX_NFTS_PER_WALLET: uint256
OWNER_NFTS: uint256
WL_CLAIM_COST: uint256
merkleRoot: bytes32
RESERVED_NFTS: uint256
reservedNFTsMinted: uint256
CLAIM_END_TIMESTAMP: uint256
dropPhases: mapping(uint256=>DropPhase)
currentDropPhase: uint256
numDropPhases: uint256
Private:
_getRandomTokenId(to: address, updatedNumAvailableTokens: uint256): uint256
Internal:
_baseURI(): string
_setTokenURI(tokenId: uint256, _tokenURI: string)
_mintRandomNFT(to: address, cost: uint256)
_burn(tokenId: uint256)
_constructTokenURI(tokenId: uint256): string
isEOA(addr: address): bool
External:
claim(merkleProof: bytes32[]) <<nonReentrant>>
mint(numTokens: uint256) <<nonReentrant>>
mintUnclaimedReservedNFTs(numTokens: uint256) <<onlyOwner>>
setDropPhase(phaseIndex: uint256, timestamp: uint256, availableNFTs: uint256, cost: uint256, discount: uint256) <<onlyOwner>>
activateDropPhase(phaseIndex: uint256) <<onlyOwner>>
getDropPhaseByID(phaseIndex: uint256): DropPhase <<onlyOwner>>
getCurrentDropPhase(): DropPhase
updateWhitelist(_root: bytes32) <<onlyOwner>>
Public:
<<event>> NFTMinted(to: address, cost: uint256, tokenId: uint256)
<<event>> DropPhaseActivated(phaseIndex: uint256)
<<event>> DropPhaseAdded(phaseIndex: uint256, timestamp: uint256, availableNFTs: uint256, cost: uint256, discount: uint256)
initialize(name: string, symbol: string, baseURI: string, receiver: address, stablecoin: address, _merkleRoot: bytes32) <<initializer>>
tokenURI(tokenId: uint256): string
supportsInterface(interfaceId: bytes4): bool
<<Interface>>
IERC721ReceiverUpgradeable
Goldrush.sol
External:
onERC721Received(operator: address, from: address, tokenId: uint256, data: bytes): bytes4
Migrations
Migrations.sol
Public:
owner: address
last_completed_migration: uint
Public:
<<modifier>> restricted()
setCompleted(completed: uint) <<restricted>>

Conclusion

The smart contracts reviewed in this audit contain no critical severity issues and all Medium to Low issues have either been corrected or acknowledged.

Please check the disclaimer above and note, the audit makes no statements or warranties on business model, investment attractiveness or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by Owner.