INFINIT3
STUDIOS

Smart Contract Audit

Prepared For:
W3bMint

Date:
12/14/22

Table of Contents

Table of Contents **2**

Summary **4**

Overview **5**

Project Summary 5
Audit Summary 5
Vulnerability Summary 5
Audit Scope 6

Findings **7**

721-001 | Set unchangeable variables as Immutable or Constant 8
721-002 | Removed unused items 9
721-003 | Declare functions as External if Public is not necessary 10
721-004 | Use 'call' instead of 'transfer' to withdraw funds 11
721-005 | Unnecessary Counters 12
721-006 | Use _totalMinted() instead of totalSupply() 13
721-007 | Efficient allowance for free and public mint 14
721-008 | Tokens not minted within phase windows will remain unminted 15
721-009 | Controls to prevent minting on same block is ineffective 16
1155-001 | Remove Unused Items 17
1155-002 | Simplify minting of multiple ERC1155 tokens. 18

Disclaimer **19**

Summary

This report was prepared to summarize the findings of the audit performed for W3bMint of their ERC-721 and ERC-1155 Contracts. The primary objective of the audit was to identify issues and vulnerabilities in the source code. This audit was performed utilizing Manual Review techniques and Static Analysis.

The audit consisted of:

- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Static Analysis utilizing Slither Static Analyzer.
- Unit testing various functions, if necessary.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Line-by-line manual review of the entire codebase by auditors.

The following report contains the results of the audit, and recommendations to guard against potential security threats and improve contract functionality.

We would like to thank the W3bmint team for their business and cooperation; their openness and communication made this audit a success.

Overview

Project Summary

Project Name	W3bMint
Lead Audtior	White Oak Kong
Description	W3bmint is a no-code platform designed to give brands the ability to launch their NFT project and access tools to grow their community. Currently they offer both ERC721 and ERC1155 contracts.
Blockchain	Ethereum
Language	Solidity
Codebase	DevProject.sol Generated1155.sol smart-contract.ts Template.sol 1155Template.sol
Commit	c7a455db2acdc4d169f1890ff6fa85db8d62adab

Audit Summary

Delivery Date	12/14/22
Audit Methodology	Static Analysis, Manual Review, Unit Testing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Resolved
Critical	0	0	0	0	0
High	4	0	0	4	0
Medium	2	0	0	2	0
Low	3	0	0	3	0
Informational	2	0	0	3	0

Audit Scope

ID	File
721	DevProject.sol, smart-contract.ts, template.sol
1155	Generated1155.sol, smart-contract.ts, 1155Template.sol

Findings

ID	Title	Category	Severity	Status
721-001	Set unchangeable variables as immutable or constant	Optimization	High	Acknowledged
721-002	Removed unused items	Optimization	Informational	Acknowledged
721-003	Declare functions as external if public is not necessary	Optimization	Low	Acknowledged
721-004	use 'call' instead of 'transfer' to withdraw funds.	Security	Medium	Acknowledged
721-005	Unnecessary Counters	Optimization	High	Acknowledged
721-006	Use _totalMinted() instead of totalSupply()	Security	Low	Acknowledged
721-007	Efficient allowance for free and public mint	Optimization	High	Acknowledged
721-008	Tokens not minted within phase windows will remain unminted.	Functional	Medium	Acknowledged
721-009	Controls to prevent minting on same block is ineffective	Security	Low	Acknowledged
1155-001	ERC1155 - address signer unused	Optimization	Informational	Acknowledged
1155-002	ERC1155 - unwieldy minting of many ERC1155s	Functional	High	Acknowledged

721-001 | Set unchangeable variables as Immutable or Constant

Category	Severity	Location	Status
Optimization	High	Multiple	Acknowledged

Description

Any variable that cannot change in the contract should be listed as either immutable or constant, depending on when the variable is set. Unchangeable variables set in a constructor should be set as immutable and those set before (AKA hardcoded) should be set as constant.

Recommendation

In DevProject.sol, all of the following can be set to constant. This will reduce gas costs by ~10%

```
uint256 public FREE_SUPPLY = 1000;
uint256 public PUBLIC_SUPPLY = 4040;
uint256 public FREE_TX_LIMIT = 1;
uint256 public PUBLIC_TX_LIMIT = 2;
uint256 public FREE_WALLET_LIMIT = 1;
uint256 public PUBLIC_WALLET_LIMIT = 2;
uint256 public FREE_PRICE = 0 ether;
uint256 public PUBLIC_PRICE = 0.07 ether;
uint256 public FREE_START_TIME = 1663672260;
uint256 public PUBLIC_START_TIME = 1663758720;
uint256 public FREE_END_TIME = 1663758660;
uint256 public PUBLIC_END_TIME = 1663845060;
```

721-002 | Removed unused items

Category	Severity	Location	Status
Optimization	Informational	Multiple	Acknowledged

Description

Unused imports, variables, and other code should be excluded to reduce contract storage size.

Contracts MerkleProof, ECDSA, and ReentrancyGuard are imported but not utilized in the contract.

Variable usedDigests is declared but unused.

Recommendation

Remove unused imports and variables from the contracts.

721-003 | Declare functions as External if Public is not necessary

Category	Severity	Location	Status
Optimization	Low	Multiple	Acknowledged

Description

Functions that are not called both internally and externally should be declared as external (or internal if appropriate) in lieu of public.

Recommendation

`reveal` should be declared as external.

721-004 | Use 'call' instead of 'transfer' to withdraw funds

Category	Severity	Location	Status
Security	Medium	Multiple	Acknowledged

Description

It is best practice to utilize a the call method to withdraw funds from a contract, as opposed to transfer.

`transfer` places a hard limit on the txn gas limit (2300). This limits the ability to withdraw funds to smart contract wallets such as gnosis safe - as they require a greater amount of gas to complete the transaction. With the increased presence of smart contract wallets and an evolving EVM - it is wise not to impose these strict limits.

Recommendation

Update the withdraw function to use call - example:

```
function newWithdraw() external onlyOwner {
    uint256 RLFee = (address(this).balance * 600)/10000;
    (bool success, ) = payable(RL_ADDRESS).call{value: RLFee}("");
    require(success, "Transfer failed.");
    (bool successTwo, ) = payable(msg.sender).call{value:
address(this).balance}("");
    require(successTwo, "Transfer failed.");
    }
```

721-005 | Unnecessary Counters

Category	Severity	Location	Status
Optimization	High	Multiple	Acknowledged

Description

The contract uses counters for tracking of global free and public mints separately. Assuming both mint periods do not occur simultaneously, this tracking is not necessary. Simply use the _totalMinted() method and compare it to the FREE_SUPPLY/MAX_SUPPLY respectively.

This modification will reduce gas fees by ~11,000 gas units (~10%) per minting transaction.

Recommendation

Replace global mint counters with the _totalMinted() method.

721-006 | Use _totalMinted() instead of totalSupply()

Category	Severity	Location	Status
Security	Low	Multiple	Acknowledged

Description

In ERC721A, totalSupply() refers to the total number of tokens minted - any burned tokens.

While this contract does not inherit an accessible burn function, it is a good practice to use _totalMinted() instead of totalSupply() when tracking minting supply. Failing to do so could result in additional tokens being minted beyond the intended maximum supply. Making this adjustment now may prevent you from incidentally introducing this possibility into future contracts that do allow burn of tokens.

Recommendation

Replace totalSupply with _totalMinted().

721-007 | Efficient allowance for free and public mint

Category	Severity	Location	Status
Optimization	High	Multiple	Acknowledged

Description

In the current contract structure, a user that mints a free NFT will only be able to mint PUBLIC_WALLET_LIMIT - numberMinted.

This may be as intended, however it potentially limits the sales during a public mint.

The contract can be modified to track free mint limits using numberMinted(), and then tracking public mint limits using the _getAux/_setAux methods available in ERC721A. This would allow a minter to mint the maximum number of free mints, and the maximum number of public mints.

_setAux() allows uint64 value to be stored and associated with a wallet address during minting in an efficient manner. _getAux() allows for the retrieval of the stored data.

Recommendation

Consider an implementation such as this:

```
function PUBLICMint(uint64 numberOfTokens) external isSecured() payable{
    require(block.timestamp >= PUBLIC_START_TIME && block.timestamp <=
PUBLIC_END_TIME, "NOT_IN_DATE_RANGE");
    require(numberOfTokens + _totalMinted() <=
MAX_SUPPLY,"NOT_ENOUGH_SUPPLY");
    require(_getAux(msg.sender) + numberOfTokens <= PUBLIC_WALLET_LIMIT,
"EXCEED__MINT_LIMIT");
    require(numberOfTokens <= PUBLIC_TX_LIMIT,"EXCEED_MINT_LIMIT");
    require(msg.value == PUBLIC_PRICE * numberOfTokens, "WRONG_ETH_VALUE");
    addressBlockBought[msg.sender] = block.timestamp;
   _setAux(numberOfTokens + _getAux(msg.sender));
    _safeMint(msg.sender, numberOfTokens);
  }
```

721-008 | Tokens not minted within phase windows will remain unminted

Category	Severity	Location	Status
Functional	Medium	Multiple	Acknowledged

Description

The contract strictly specifies that minting functions can only be called during the time gated windows controlled by the individual phase start/end times. This means that any supply reserved for a certain phase will be unable to be minted if it is not done so during the designated phase.

For example:

FREE_SUPPLY = 1000
PUBLIC_SUPPLY = 1000
MAX_SUPPLY = 2000

If only 900 tokens are minted during the FREE phase of minting, a maximum of 1900 tokens can be minted during both phases combined, regardless of the MAX_SUPPLY.

Recommendation

Suggestions to address the above include the following:

1. Replace timestamps will bool operators that allow for manual toggling of phase.
2. Leave the final phase without an ending time, and allow for tokens to be minted up to MAX_SUPPLY.
3. Track token minting using one global counter (_totalMinted()) instead of separate counters, and ensure that the final phase supply is == to MAX_SUPPLY.

721-009 | Controls to prevent minting on same block is ineffective

Category	Severity	Location	Status
Security	Low	Multiple	Acknowledged

Description

The contract uses a control through the modifier isSecured() to stop individual addresses from processing multiple mints on the same block: require(addressBlockBought[msg.sender] < block.timestamp, "CANNOT_MINT_ON_THE_SAME_BLOCK");

This is technically fine, and will prevent people from minting multiple times in the same block; however, it will prevent non-bot users from minting multiple times quickly, and could result in failed transactions that affect UX. Additionally, the cost of tracking this is gas intensive.

Many bots operating in the NFT market today are minting using multiple wallets simultaneously - not minting multiple transactions from the same wallet. The control put in place is not going to stop this - captcha and signatures can work, but are also not foolproof.

Recommendation

Remove the mapping addressBlockBought.

1155-001 | Remove Unused Items

Category	Severity	Location	Status
Optimization	Informational	Multiple	Acknowledged

Description

Unused imports, variables, and other code should be excluded to reduce contract storage size.

`Address signer` is declared but unused.

Recommendation

Remove `Address signer.`

1155-002 | Simplify minting of multiple ERC1155 tokens.

Category	Severity	Location	Status
Functional	High	Multiple	Acknowledged

Description

The current mint function structure in the ERC1155 contract is set up to use a different function for each tokenID. For example, if there are 20 different ERC1155s that can be minted, there will be 20 separate mint functions.

Recommendation

I would update this logic to include just two mint functions (one presale and one public), and add variables to mint different tokenIDs within the function. Example:

```
function Mint(uint256 numberOfTokens, bytes32[] memory proof, uint256
tokenId) external isSecured() payable{
    require(MerkleProof.verify(proof, MerkleRoots[1],
keccak256(abi.encodePacked(msg.sender))),"PROOF_INVALID");
    require(numberOfTokens + totalSupply(tokenId) <= SupplyTracker[1],
"NOT_ENOUGH_SUPPLY");
    require(balanceOf(msg.sender, tokenId) + numberOfTokens <=
WalletLimit[1],"EXCEED__MINT_LIMIT");
    require(numberOfTokens <= TxLimit[1], "EXCEED_MINT_LIMIT");

    addressBlockBought[msg.sender] = block.timestamp;
    _mint(msg.sender, tokenId,numberOfTokens, '');
  }
```

Disclaimer

The audit performed by Infinit3 Studios is intended to identify function weaknesses, potential security threats, and improve performance. It does not provide a guarantee of contract performance or the absence of any exploitable vulnerabilities. Any alterations to the source code provided could potentially invalidate the analysis provided in this audit report. Infinit3 Studios is not liable for any losses or damages incurred as a result of contract deployment and the proceeding occurrences.