Wristables

Security Audit

Mar 28th 2022

Version 1.1.0 (Including fixes)

Version 1.0.0 was released at 22nd March, link
This version includes review of fixes as well.

Opening Statement

This document includes the results of the security audit for Wristables smart contracts. The security audit was performed by the Shipyard audits team from Mar 9th, 2022 to Mar 21st, 2022.

We checked upgradability, checked various mint methods, analysed how new batches can be made available, how payments can be splitted, and other exposed features.

And couldn't find any issues with high severity that would substantially compromise the integrity of the project.

We have identified 1 medium level and 1 low-level issue,
We also found some substantial gas optimizations.

Let's Start !

Disclaimer: While Shipyard's review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Specification

Our understanding of the specification was based on the following sources:

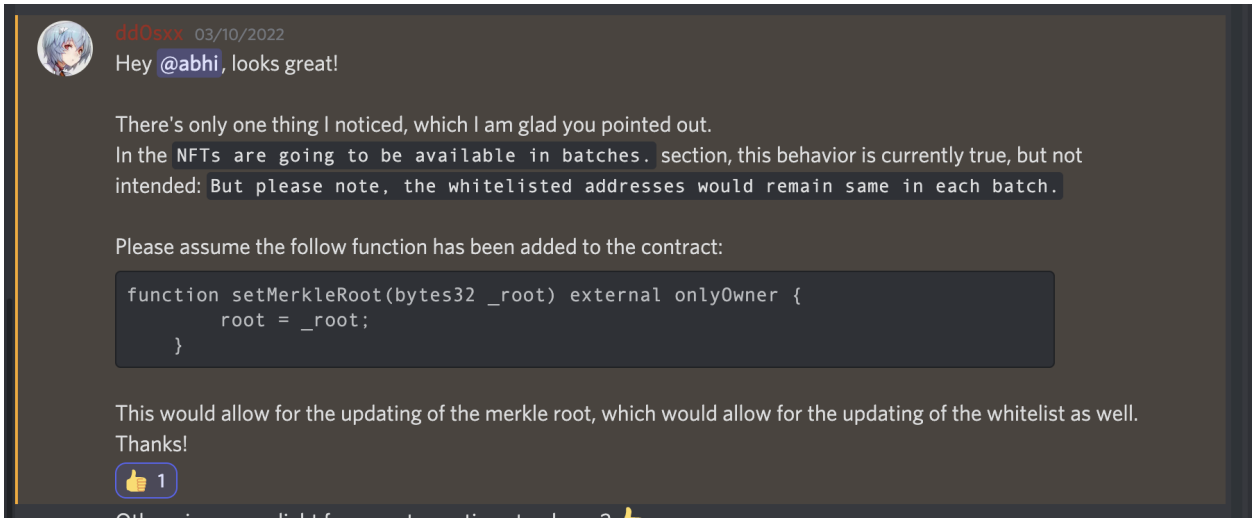
- Wristables Project Scope
- Discussions with the Wristables team
- Comments in the contract source code / README

Source Code

The following source code was reviewed during the audit:

Repository	Commit
Github	58c727478c3160b60c2fa0f0bd75e6e7f9c1e35b

Plus we considered the following addition to the contract, as per the request of the wristables team.



Link

Specifically, we audited the following contracts:

Contract	Sha256
Wristables.sol	1a4d505c7d2000026a3c4946ee6cdad6b7d3db7300cd8084d98f44448bbbf5fd

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Security Issues and Recommendations

Severity Level Reference

Level	Description
High	The issue poses existential risk to the project, and the issue identified could lead to massive financial or reputational repercussions.
Medium	The potential risk is large, but there is some ambiguity surrounding whether or not the issue would practically manifest.
Low	The risk is small, unlikely, or not relevant to the project in a meaningful way.
Code Quality	The issue identified does not pose any obvious risk, but fixing it would improve overall code quality, conform to recommended best practices, and perhaps lead to fewer development issues in the future.

[Issue-1]

MEDIUM

Fixed in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9,
by creation of a new function called setAllSaleParams().

Depending on order of transactions, NFTs could be minted at 0 price and could be distributed unfairly upto certain limit.

Wristables NFT are going to be available in batches,
so once a new batch is going to be made available,
a set of functions are needed to be called

They are,

```
setAvailableSupply()
setMerkleRoot(new root) let's say whitelist is changed
setIndexWl()
setSaleActive(false)
setMintPrice(new price)
```

Following are the various paths depending on order.

```
Path 1
1                       2
setAvailableSupply(x) => setMintPrice(y eth)

Between 1 and 2
Free Mint: Users* can mint for 0 price before setMintPrice() is
finalised for the first batch, and can mint for prev batch price for
subsequent batches.
```

*(Only WL address if setSaleActive(false) is finalised first, if not any address)

Path 2

```
1                        2
setAvailableSupply() => setSaleActive(false)
```

Between 1 and 2
Unfair Distribution: Anyone can mint for new batch not only whitelisted addresses

Path 3

```
1                       2                            3
setAvailableSupply(x)=> setMerkleRoot(new root) => setIndexWl(prev +1)
```

Between 2 and 3
Unfair Distribution: New WL addresses as per this new root, can claim 2 NFTs
One NFT of this batch, and after 3, one more.

Path 4

```
1                       2                        3
setAvailableSupply(x)=> setIndexWl(prev +1) => setMerkleRoot(new root)
```

Between 2 and 3
Unfair Distribution: Old WL addresses of prev batch, can claim NFT of this new batch before 3.

With MEV getting traction with respect to NFT mints, we suggest you consider resolving this issue. [tweet 1], [tweet 2]

Consider:

Solution 1

Calling `setAvailableSupply()` always at last once everything else is finalised,
so even if someone tries to take advantage, it will revert, as they won't have anything to mint from contracts.

Solution 2

The best possible solution is to make these individual separate transactions, one atomic transaction.
We can achieve so either by writing one wrapper method in contract, or using `multicall()`.
This will cost low on gas as well, as we would have to pay fixed gas price only once, not for 5 times.
`Saving 21000 * 4 in gas units for each batch movement.`

[Issue-2]

~~LOW~~

Fixed in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9,
by adjusting the `require` condition. L75
Please note that in revised version,
`airdrop()` is removed and its functionality is made available through `batchAirdrop()` only.

Due to logical error in `airdrop()` and `batchAirdrop()`, the owner can not airdrop the last NFT of a given batch.

Via both of these functions, the airdrop can only be done upto `Id => MAX_SUPPLY - 1` at last, or only upto the `availableSupply - 1` for a given batch.

Please note as per conversation with the Wristables team, `MAX_SUPPLY` represents the max token id possible, and `availableSupply` represents max token id available for mint

this batch.
It's not with respect to quantity.
So if availableSupply is 9999, then 0-9999 are the available token ids, and a total of 10000 NFTS are there.

For example,
Consider airdrop (to, quantity) being called with following states.

```
79      /// @dev sends the next token to the `to` address for free + gas
80      function airdrop (address to, uint256 quantity) public payable onlyOwner nonReentrant {
81          uint mintIndex = _tokenSupply.current();
82          require(mintIndex + quantity <= availableSupply, "exceeds available supply");
83          for (uint256 i = 0; i < quantity; i++) {
84              _safeMint(to, mintIndex + i);
85              _tokenSupply.increment();
86          }
87      }
88
```

State 1
availableSupply = 9999 (max value)
mintIndex = 0 and quantity = 9999

Would only mint upto 9998

State 2
availableSupply = 9999
mintIndex = 0 and quantity = 10000

Would revert with "exceeds available supply"

State 3
availableSupply = 9999
mintIndex = 9999 and quantity = 1

Would revert with "exceeds available supply"

same issue is with **batchAirdrop()**

Consider:
Correcting the logic on L82 with
`require(mintIndex + quantity <= availableSupply + 1, "exceeds available supply");`
And on L92
`require(mintIndex + to.length <= availableSupply + 1, "exceeds token supply");`

There was one more issue with wristables initial version, it was with `supportInterface()` being implemented incorrectly,
This was observed in another independent audit done by the Shipyard only,
and is fixed in this revised version.
135c10a61f9698f1d0df17af8cad1e9b885b4be9.

Gas Optimisations

1) Claimed whitelisted addresses for a given batch are tracked by claimedWL mapping.

```
mapping(address => bool[10]) public claimedWL;
// stores addresses that have claimed whitelisted tokens, set to fixed array because a dynamic array inside of a mapping does not fill with falsey values by default. There won't be more than 10 drops so this is safe for us to assume.
```

Hence, it is a combination of two keys which indicate if the user has claimed or not.
`(address, indexWL) => (true, false)`

We can leverage hashing here, combining this two keys
`hash(address, indexWL) => (true, false)`

We do this, and we would need only 1 slot per user to store the details.

Optimised in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9

2) We can pack following variables, saving us gas cost with respect to SLOAD and SSTORE

Original : 6 slots

```
slot 1
uint256 public availableSupply;
slot 2
uint256 public MAX_SUPPLY;
slot 3
uint256 private mintPrice;
slot 4
bool private toggleAuction;
bool private saleActive;
slot 5
bytes32 public root;
slot 6
uint32 public indexWL;
```

Optimised : 2 slots

```
slot 1
2: uint16 public availableSupply; // safe as maxValue: 9999 < 2^16-1 (65,535)
2: uint16 public MAX_SUPPLY; // safe as maxValue: 9999 < 2^16-1 (65,535)
1: uint8 public indexWL; // safe as maxValue: 10 < 2^8-1 (255)
1: bool private toggleAuction;
1: bool private saleActive;
16 : uint128 private mintPrice;
// as per us mintPrice realistically cant go beyond uint128 =
// 340,282,366,920,938,463,463,374,607,431,768,211,455 wei

Total : 28 < 32 => so one slot

slot 2
bytes32 public root;
```

Please note if you decide to do this change,
Pay caution to casting safely and overflow issues while doing arithmetic on them, as it will be done in that scope only.

Optimised in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9

3) We can make `L25: MAX_SUPPLY immutable`, as it's being set only once. We do this and we save `SLOAD` while reading it everywhere.

Optimised in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9
***Made constant instead of immutable.**

4) We can pack variables of DutchAuction struct : L39

Original : 5 slots

```
struct DutchAuction {
        uint256 startingPrice;
        uint256 floorPrice;
        uint256 startAt;
        uint256 expiresAt;
        uint256 priceDeductionRate;
}
```

Optimised : 2 slots

```
struct DutchAuction {
        // slot 1
        16 : uint128 startingPrice;
        keep it same as of mintPrice
        16 : uint128 floorPrice;
        keep it same as of mintPrice
        // slot 2
        8: uint64 startAt;
        8: uint64 expiresAt;
        To be in context why this is safe
        Max value of uint64 => 18,446,744,073,709,551,615
        Date and time (GMT): 21 March 9999 3:06:51 PM
        Timestamp in milliseconds: 253,377,644,811,000
        16: uint128 priceDeductionRate;
        keep it same as of mintPrice
}
```

Please note if you decide to do this change
Pay caution to casting safely and overflow issues while doing arithmetic on them, as it will be done in that scope only.

Optimised in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9

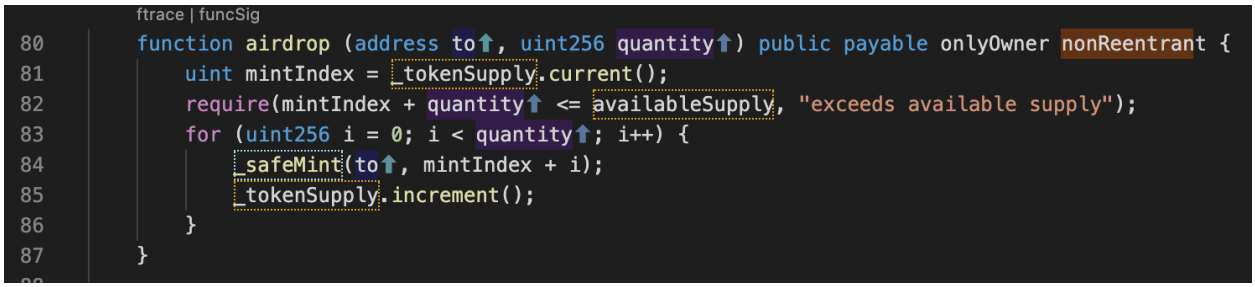
5) It is true that using `safeMint()`, allows user to reenter using hook, but though they can re enter they can't exploit as

- For redeem(), would revert on L127, due to claimedWL Mapping. check on L127, effect on L128, interaction at L129
- For airdrop() and batchAirdrop(), would fail as msg.sender won't be owner
- For mintAuction() and mint(), would fail as msg.value would be 0, so you need to pay only again.

If you also agree with our analysis, please consider removing the nonReentrant modifier for above functions, saving `SLOAD` and `SSTORE`.

Optimised in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9

6) Counters library from OpenZeppelin is used to increment ids, due to this we are restricted to increment one by one only.



```
ftrace | funcSig
80      function airdrop (address to⬆, uint256 quantity⬆) public payable onlyOwner nonReentrant {
81          uint mintIndex = _tokenSupply.current();
82          require(mintIndex + quantity⬆ <= availableSupply, "exceeds available supply");
83          for (uint256 i = 0; i < quantity⬆; i++) {
84              _safeMint(to⬆, mintIndex + i);
85              _tokenSupply.increment();
86          }
87      }
```

Same applies to `batchAirdrop()` as well

```
We have to do ++, x times,
Where we could have done
tokenSupply = tokenSupply + x
```

We can do the above ad hoc update, if we write our own logic for incrementing tokenSupply, which should not be a concern, as logic being inherited from counters is not complex.

Optimised in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9

7) Consider defining `initialize()`, `airdrop()`, `batchAirdrop()`, `tokenURI()` external from public.

Optimised in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9

Code Quality

1) L4, redundant import : `import "hardhat/console.sol";`
L15, redundant declaration : `using AddressUpgradeable for address;`

We agree at the end this won't be present in bytecode, as the compiler would do tree-shaking, but still removing them is cleaner in our opinion.

Considered in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9

2) L22, L26, L27, L28, variables declared on these lines are private
As we know, making variables private is not going to stop attackers from accessing those values, then why stop UIs to access it directly using abi, or some contract to build on top of it.

Considered in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9

Code Coverage

Code coverage was measured by running **solidity-coverage** at the root of the project.

File	%Stmts	%Branch	%Funcs	%Lines	Uncovered Lines
Wristables.sol	87.72	58.82	85.71	87.93	..207,208, 209
All Files	87.72	58.82	85.71	87.93	

Evaluation

The code coverage for the wristables repository could be improved in our opinion.
In order to improve it consider:

1. Testing revert cases of following requires.
   L92, L102, L109, L116, L117, L125, L126, L127, L135, L174
2. Testing setBaseURI(), setIndexWL(), tokenURI().

Automated Analysis

Slither

We ran slither, and analysed its report, and was unable to find anything of concern.
The whole Slither report can be found here.
While we paid close attention to the ones highlighted in bold, we ultimately found them to be false positives and not actual issues based on our own understanding of the intended behaviour and code, as well as conversations with the Wristables development team

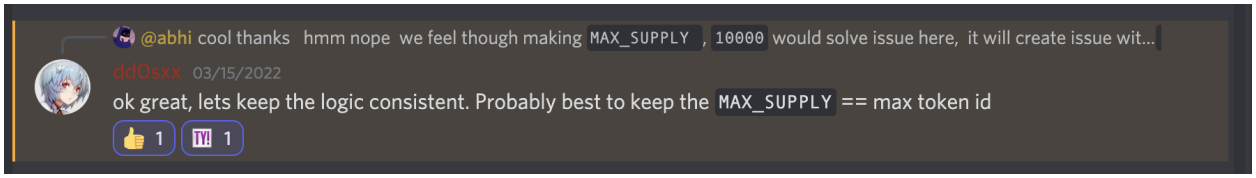
Appendix

Exhibit A - Suggested Renamings, Refactors and Cleanups

1) Consider renaming `toggleAuction to isAuctionActive`
   Similarly for `setToggleAuction()`, consider renaming to `setAuctionStatus()`

2) Initially, we were deducting `MAX_SUPPLY`, in units of quantity, similar to `TotalSupply.`
   but later, once we discussed with Wristables team, we got to know, that `MAX_SUPPLY` is being meant as max token id.
   Please consider renaming it to `MAX_TOKEN_ID` only, so there is no scope for confusion.
   Same for availableSupply => availableTokenId

   Considered in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9



3) Consider renaming the SaleActive() modifier to `isSaleActive()` `[mixed-case]`.

Exhibit B - Comment and Documentation Typos and Potential Improvements

1) L55: Comment is no more applicable
   ```
   /// @dev `maxBatchSize` refers to how much a minter can mint at a time.
   ```
2) L62: Redundant commented console statement
3) L76: Incomplete comment
4) L198: Wrong comment

All considered in revised version : 135c10a61f9698f1d0df17af8cad1e9b885b4be9

Exhibit C - Questions raised and Responses

1) Division is done before multiplication in dutchAuction-deduction calculation ?
   Response: Its intended, as for one slot, price should be same
2) Refund on excess ETH paid by a user in dutchAuction ?
   Response: Not in scope
3) Can availableSupply be increased before the last batch is completely minted ?
   Response: No

Exhibit D -Disclaimer

We makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and We specifically disclaims all implied warranties or merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

We will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand against company by any other party. In no event will we be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if we has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Wristables team and only the source code we note as being within the scope of our review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than us. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that we are not responsible for the content or operation of such websites, and that we shall have no liability to your or any other person or entity for the use of third party websites. We assume no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.