Elder Gods Smart Contracts Audit

September, 2022

Oleksandr Zadorozhnyi, Oleh Rubanik, Yaroslav Tumanov

1. **INTRODUCTION.** Elder Gods requested 4irelabs a review of the contract implementing their NFT token.

   There is 1 contract source code in scope provided in the archive file.

2. **WARRANTY.** Audit is provided on an "as is" basis, without warranty of any kind, express or implied. The Auditor does not guarantee that the Code Review will identify all instances of security vulnerabilities or other related issues.

3. **EXECUTIVE SUMMARY.** 4 Critical, 9 Major and 17 Low/Note issues have been found. Contract owner has the superpower to stop rent (5.1.35, 5.1.37, 5.1.38). Contract is not fully ERC4709 compliant (5.1.39). There are no known compiler bugs, for the specified compiler version, that might affect the contracts logic.

4. **CRITICAL BUGS AND VULNERABILITIES.** 4 Critical (5.1.17, 5.1.26, 5.1.30, 5.1.31), 9 Major (5.1.10, 5.1.12, 5.1.13, 5.1.18, 5.1.32, 5.1.35, 5.1.37, 5.1.38, 5.1.39) issues have been found and described below.

5. LINE BY LINE REVIEW.

5.1. ElderGODS.sol:

5.1.1. Line 1295. Gas optimization: Mapping *(uint256 => TokenType) public tokenType* could be updated to *(uint256 => bool) public isSloth*, then getter *isSloth()* would not be needed and code would be more clear.

5.1.2. Line 1297. Note: typo in *'InfniGods'*.

5.1.3. Line 1298. Low: variable *mintedInPublic* never used.

5.1.4. Line 1399. Note: immutable *MAX_SUPPLY* could be constant for more clear code.

5.1.5. Line 1304. Gas optimization: variable *slothAirdropCount* could be deleted since it is never used for reading only incrementing on line 1331.

5.1.6. Line 1308. Gas optimization: *variable sale_start_time* should be constant.

5.1.7. Line 1309. Gas optimization: *variable sale_end_time* should be constant.

5.1.8. Line 1316. Note: The name and symbols of the token should be specified by the parameter.

Recommendation: Pass the name and symbol as a constructor parameter.

5.1.9. Line 1326, 1395 Low: The gas calculation logic is too expensive.

Recommendation: Change the gas subtraction logic to an easier one. The function should be passed a parameter that will serve as the gas limit. In the condition, you should use this parameter to compare it with the rest of the gas. (*gasleft() > gasLimit*). This will help not to declare unnecessary variables and make the gas logic more flexible.

5.1.10. Line 1328, 1345. Major: The loop will be infinite if the condition is not met on the first call.

Recommendation: Use the counter outside of the condition and save it in the contract storage. The counter will display the id of the end of the cycle and the next call will start from the counter.

5.1.11. Line 1328 and next `for` loops. Gas optimization: `for` loop could have more optimized flow, with prefix increment ++i in unchecked block, like next:

```
for (uint i = 0; i < length;) {
        // Do stuff
unchecked {
++i;
        }
    }
```

5.1.12. Line 1333, 1350, 1389, 1398. Major: Using *_safeMint()* function could lead to reentrancy vulnerability.

Recommendation: use reentrancy guard modifier from OpenZeppelin library in functions that call *_safeMint().*

5.1.13. Line 1324, 1341,1393. Major: Hardcoded value for maximum used

gas could lead to `denial of service` bugs.

Recommendation: Add parameter *maxGas* in functions *airdropSloth()*, *airdropStandard()*, *returnToTreasury()* instead of hardcoded value 5000000.

5.1.14. Line 1358. Low: The Mint feature is too expensive. When passing a long array as parameters, the function may fail due to lack of gas to perform all operations.

Recommendation: Warn the user that the function may fail if an array that is too large is passed.

5.1.15. Line 1364, 1369 Gas optimization: Using error messages that have length less than 32 bytes could save gas in revert cases. Also the error text is not informative.

5.1.16. Line 1371-1387. Gas optimization: flow on these lines could be updated to more efficient without `while` loop:

```
for (uint256 y; y < _tokenId.length; y++) {
            if (mintedAmount[_tokenId[y]] < 2) {
                if (count >= 2) {
                    count -= (2 - mintedAmount[_tokenId[y]]);
                    mintedAmount[_tokenId[y]] = 2;
                } else {
                    mintedAmount[_tokenId[y]] += count;
                    break;
                }
            }
        }
```

5.1.17. Line 1397. **Critical:** Due to non-strict(!) comparing *totalSupply() <= MAX_SUPPLY* there is a possibility that a token with id 10001 would be minted.

Recommendation: Update comparison sign to strict "<".

5.1.18. Line 1398. Major: Mint for zero address. If the treasury address was not originally set, this function will mint to address zero.

Recommendation: Check that treasury address not equal to zero.

5.1.19. Line 1409, 1413, 1519. Note: Function with visibility *public* has never been used in this contract.

Recommendation: Change visibility to *external*.

5.1.20. Line 1418. Gas optimization: No needed branch *`else`* since all values in array *result* already initialized with 0 value.

5.1.21. Line 1434. Note: Mixed style of functions and variables location that brokes solidity Style Guide:
https://docs.soliditylang.org/en/v0.8.16/style-guide.html#order-of-layout

5.1.22. Line 1437. Low: *godsFee* variable better to make public, so external users would be able to read it.

5.1.23. Line 1438. Note: The immutable variable denominator is never reassigned.

Recommendation: Change the immutable variable to a constant.

5.1.24. Line 1450. Gas optimization: *RentBids* structure not optimized.

Recommendation: Change structure to: {uint256, uint256, address, uint64, address, uint64}

Explanation:
https://docs.soliditylang.org/en/v0.8.13/internals/layout_in_storage.html

5.1.25. Line 1459. Low: using extending array *rentBids* would lead to denial of service for *getBidsForNFT()* and *getActiveBids()* functions in future.

Recommendation: Instead of array use mapping with *RentBids* structure as values and additional uint256 *counterOfBids* as keys. Delete functions *getBidsForNFT()* and *getActiveBids()*, such data collection flow should be provided externally.

5.1.26. Line 1485. Critical: Due to check for NFT owner inside *_setUser()* function *acceptRent()* function would be broken since on line 1521 there is check that *msg.sender* is not the NFT owner.

Recommendation: Move check on line 1485 inside *setUser() public* function.

5.1.27. Line 1488. Low: Event *UpdateUser()* would be emitted only in case if *setUser()* function called, but not in case if internal function *_setUser()* called.

Recommendation: Move event *UpdateUser()* emitting inside *_setUser()* internal function.

5.1.28. Line 1501. Low: Missed event emit *NFTListedForRent*.

5.1.29. Line 1516. Low: Emitting event *NFTListedForRent()* during rent delete could mislead external listeners. Recommendation: Emit new event *NFTDeletedForRent().*

5.1.30. Line 1519. **Critical:** After NFT transfer the previous owner could accept rent (with low price/long duration) that he or she put before transfer(selling). This would lead to the inability of the new owner to set a new user.

Recommendation: Add *lender* field to *RentableItems* struct and inside *acceptRent()* function check that current *owner* is a *lender*. Inside the *putForRent()* function set l*ender* value by current *owner* address.

5.1.31. Line 1527, 1528, 1530, 1568, 1569, 1571. **Critical:** Some ERC20 not reverts on fail transfers, instead of reverting they returns *false* value. This could lead to situations when malicious users could rent NFTs without paying rent funds.

Recommendation: use safeERC20 library from OpenZeppelin for token transfers. Additionally use reentrancy guard modifier to prevent reentrancy in *safeTransferFrom* calls.

5.1.32. Line 1528, 1569. Major: In case when price of rent and godsFee have low values - *toTreasury* fee amount could have 0 value. Some ERC20 tokens could revert 0 transfers, which would break such rent transactions.

Recommendation: Add checks for *toTreasury* fee amount to be greater than 0 before transfer.

5.1.33. Line 1539. Note: No needed check since user balance could change by the time of renting.

5.1.34. Line 1556. Low: Emitting event *BidPlaced()* during rent delete could mislead external listeners.

Recommendation: Emit new event *BidDeleted()*.

5.1.35. Line 1614. Major: Owner has superpower to break renting functionality by setting all *allowedTokens* to *false*.

5.1.36. Line 1620, 1624, 1628. Low: Missing events on sensitive changes.

5.1.37. Line 1624. Major. Malicious owner could frontrun user transitions with *acceptBid()* and *acceptRent()* calls with setting fee to 100% and taking all rent funds. Also setting godsFee to value > 1000 would break renting functionality.

Recommendation: Add *maxFee* constant which would restrict maximum value for *godsFee* and add timelock functionality on *changeRentFee()* function, so owner would not be able to increase fees instantly.

5.1.38. Line 1628. Major: Owner has superpower to break renting functionality by setting all *canRent* to *false*.

5.1.39. Line 1643. Major : Non-compliance with the standard EIP-4907. *supportsInterface*.

Recommendation: Add the missing features to the contract. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-4907.md

5.2. Documentation and contracts code mismatches:

5.2.1. Provided documentation contains restintion numbers of NFTs that are not present in contract code from the scope: 260 SLOTH GODS, 8000 GODS reserved for Community Pass, 518 GODS reserved for Ultimate Pass holders.

5.2.2. Documentation contains information about airdrop, public, private sales and mint for treasury. While due to contract code NFTs could be minted in “airdrop” and “mint for treasury” functions by owner and “mint” function by users. There is no 5% restriction for treasury mint and no dividing for public and private mint.

5.2.3. Due to documentation - sale should continue for 7 days, while in code start and end time values have bigger difference.