Art Gobblers Security Review

Auditors

Kurt Barry, Lead Security Researcher

Leo Alt, Lead Security Researcher

Hari, Lead Security Researcher

Emanuele Ricci, Security Researcher

Patrickd, Security Researcher

Hrishikesh Bhat, Apprentice

Devansh Batham, Apprentice

Alex Beregszaszi, Consultant

Report prepared by: Pablo Misirov, Devansh Batham & Hrishikesh Bhat

October 25, 2022

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Art Gobblers is an NFT art factory & gallery owned by an exclusive group of users. Gobblers are animated characters with generative attributes on which project users can draw, feed art to & generate tokens. The Art Gobblers project aims to become a nexus point for artists, collectors, and crypto enthusiast over time.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Art Gobblers according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 12 days in total, Art Gobblers engaged with Spearbit to review Art-Gobblers. In this period of time a total of 23 issues were found.

Summary

Project Name	Art Gobblers
Repository	art-gobblers
Commit	fe647c8a7e45...
Type of Project	Art, NFT
Audit Timeline	July 4th - July 15th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	3
Gas Optimizations	4
Informational	16
Total Issues	23

5 Findings

5.1 Low Risk

5.1.1 The `claimGobbler` function does not enforce the `MINTLIST_SUPPLY` on-chain

Severity: *Low Risk*

Context: ArtGobblers.sol#L287-L304

Description: There is a public constant `MINTLIST_SUPPLY` (2000) that is supposed to represent the number of gobblers that can be minted by using merkle proofs. However, this is not explicitly enforced in the `claimGobbler` function and will need to be verified off-chain from the list of merkle proof data.

The risk lies in the possibility of having more than 2000 proofs.

Recommendation: Consider implementing the following recommendations.

1. The check can be enforced on-chain by adding a state variable to track the number of claims and revert if there are more than `MINTLIST_SUPPLY` number of claims. However, this adds a state variable which needs to be updated on each claim, thereby increasing gas costs. It is understandable if such suggestion is not implemented due to gas concerns.
2. Alternatively, publish the entire merkle proof data so that anyone can verify off-chain that it contains at most 2000 proofs.

ArtGobblers: Documented this in ArtGobblers.sol#L335.

```
    /// @dev Function does not directly enforce the MINTLIST_SUPPLY limit for gas efficiency. The
    /// limit is enforced during the creation of the merkle proof, which will be shared publicly.
```

5.1.2 Feeding a gobbler to itself may lead to an infinite loop in the off-chain renderer

Severity: *Low Risk*

Context: ArtGobblers.sol#L653

Description: The contract allows feeding a gobbler to itself and while we do not think such action causes any issues on the contract side, it will nevertheless cause potential problems with the off-chain rendering for the gobblers.

The project explicitly allows feeding gobblers to other gobblers. In such cases, if the off-chain renderer is designed to render the inner gobbler, it would cause an infinite loop for the self-feeding case.

Additionally, when a gobbler is fed to another gobbler the user will still own one of the gobblers. However, this is not the case with self-feeding,.

Recommendation: Consider implementing the following recommendations.

1. Disallowing self-feeding.
2. If self-feeding is allowed, the off-chain renderer should be designed to handle this case so that it can avoid an infinite loop.

ArtGobblers: Fixed in commit fa81257.

Spearbit: Acknowledged.

5.1.3 The function `toString()` does not manage memory properly

Severity: *Low Risk*

Context: LibString.sol#L7-L72

Description: There are two issues with the `toString()` function:

1. It does not manage the memory of the returned string correctly. In short, there can be overlaps between memory allocated for the returned string and the current free memory.
2. It assumes that the free memory is clean, i.e., does not explicitly zero out used memory.

Proof of concept for case 1:

```
function testToStringOverwrite() public {
    string memory str = LibString.toString(1);

    uint freememptr;
    uint len;
    bytes32 data;
    uint raw_str_ptr;

    assembly {
        // Imagine a high level allocation writing something to the current free memory.
        // Should have sufficient higher order bits for this to be visible
        mstore(mload(0x40), not(0))
        freememptr := mload(0x40)
        // Correctly allocate 32 more bytes, to avoid more interference
        mstore(0x40, add(mload(0x40), 32))
        raw_str_ptr := str
        len := mload(str)
        data := mload(add(str, 32))
    }
    emit log_named_uint("memptr: ", freememptr);
    emit log_named_uint("str: ", raw_str_ptr);
    emit log_named_uint("len: ", len);
    emit log_named_bytes32("data: ", data);
}
```

```
Logs:
  memptr: : 256
  str: : 205
  len: : 1
  data: : 0x31000000000000000000000000000000000000ffffffffffffffffffffffffff
```

The key issue here is that the function allocates and manages memory region [205, 269) for the return variable. However, the free memory pointer is set to 256. The memory between [256, 269) can refer to both the string and another dynamic type that's allocated later on.

Proof of concept for case 2:

```
function testToStringDirty() public {
    uint freememptr;
    // Make the next 4 bytes of the free memory dirty
    assembly {
        let dirty := not(0)
        freememptr := mload(0x40)
        mstore(freememptr, dirty)
        mstore(add(freememptr, 32), dirty)
        mstore(add(freememptr, 64), dirty)
        mstore(add(freememptr, 96), dirty)
        mstore(add(freememptr, 128), dirty)
    }
    string memory str = LibString.toString(1);
    uint len;
    bytes32 data;
    assembly {
        freememptr := str
        len := mload(str)
        data := mload(add(str, 32))
    }
    emit log_named_uint("str: ", freememptr);
    emit log_named_uint("len: ", len);
    emit log_named_bytes32("data: ", data);
    assembly {
        freememptr := mload(0x40)
    }
    emit log_named_uint("memptr: ", freememptr);
}
```

```
Logs:
  str: 205
  len: : 1
  data: : 0x31ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
  memptr: : 256
```

In both cases, high level solidity will not have issues decoding values as this region in memory is meant to be empty. However, certain ABI decoders, notably Etherscan, will have trouble decoding them.

Note: It is likely that the use of `toString()` in ArtGobblers will not be impacted by the above issues. However, these issues can become severe if LibString is used as a generic string library.

Recommendation: For the first case, we recommend allocating 160 bytes rather than 128 bytes in LibString.sol#L22.

```
-  mstore(0x40, add(str, 128))
+  mstore(0x40, add(str, 160))
```

For the second case, the easiest fix would be to zero out the memory that follows the final character in the string.

ArtGobblers: Fixed in Solmate.

Spearbit: Acknowledged.

5.2 Gas Optimization

5.2.1 Consider migrating all `require` statements to `Custom Errors` for gas optimization, better UX, DX and code consistency

Severity: *Gas Optimization*

Context: ArtGobblers.sol, SignedWadMath.sol, GobblersERC1155B.sol, PagesERC721.sol

Description: There is a mixed usage of both `require` and `Custom Errors` to handle cases where the transaction must revert.

We suggest replacing all `require` instances with `Custom Errors` in order to save gas and improve user / developer experience.

The following is a list of contract functions that still use `require` statements:

- `ArtGobblers` `mintLegendaryGobbler`
- `ArtGobblers` `safeBatchTransferFrom`
- `ArtGobblers` `safeTransferFrom`
- `SignedWadMath` `wadLn`
- `GobblersERC1155B` `balanceOfBatch`
- `GobblersERC1155B` `_mint`
- `GobblersERC1155B` `_batchMint`
- `PagesERC721` `ownerOf`
- `PagesERC721` `balanceOf`
- `PagesERC721` `approve`
- `PagesERC721` `transferFrom`
- `PagesERC721` `safeTransferFrom`
- `PagesERC721` `safeTransferFrom` (overloaded version)

Recommendation: Consider replacing all instances of the `require` statement with `Custom Errors` across the codebase.

5.2.2 Minting of Gobbler and Pages can be further gas optimized

Severity: *Gas Optimization*

Context: Pages.sol#L126-L143, ArtGobblers.sol#L310-L331

Description: Currently, in order to mint a new Page or Gobbler users must have enough `$GOO` in their `Goo` contract balance. If the user does not have enough `$GOO` he/she must call `ArtGobblers.removeGoo(amount)` to remove the required amount from the Gobbler's balance and mint new `$GOO`. That `$GOO` will be successively burned to mint the `Page` or `Gobbler`.

In the vast majority of cases users will never have `$GOO` in the `Goo` contract but will have their `$GOO` directly stacked inside their `Gobblers` to compound and maximize the outcome.

Given these premises, it makes sense to implement a function that does not require users to make two distinct transactions to perform:

- mint `$GOO` (via `removeGoo`).
- burn `$GOO` + mint the Page/Gobbler (via `mintFromGoo`).

but rather use a single transaction that consumes the `$GOO` stacked on the `Gobbler` itself without ever minting and burning any `$GOO` from the `Goo` contract. By doing so, the user will perform the `mint` operation with only one transaction and the gas cost will be much lower because it does not require any interaction with the `Goo` contract.

Recommendation: Consider evaluating the possibility to allow users mint `Gobbler's` and `Page's` directly from the `$GOO` balance in `ArtGobbler` to save gas.

Art Gobblers: Recommendation implemented in PR 113.

Spearbit: Acknowledged.

5.2.3 Declare `GobblerReserve` `artGobblers` as `immutable`

Severity: *Gas Optimization*

Context: GobblerReserve.sol#L17

Description: The `artGobblers` in the `GobblerReserve` can be declared as immutable to save gas.

```
- ArtGobblers public artGobblers;
+ ArtGobblers public immutable artGobblers;
```

Recommendation: Declare the `artGobblers` variable as `immutable`.

Art Gobblers: Recommendation implemented in the PR 111.

Spearbit: Acknowledged.

5.3 Informational

5.3.1 Neither `GobblersERC1155B` nor `ArtGobblers` implement the ERC-165 `supportsInterface` function

Severity: *Informational*

Context: ArtGobblers.sol, GobblersERC1155B.sol

Description: From the EIP-1155 documentation:

> *Smart contracts implementing the ERC-1155 standard MUST implement all of the functions in the `ERC1155` interface. Smart contracts implementing the ERC-1155 standard MUST implement the ERC-165 `supportsInterface` function and MUST return the constant value `true` if `0xd9b67a26` is passed through the `interfaceID` argument.*

Neither `GobblersERC1155B` nor `ArtGobblers` are actually implementing the ERC-165 `supportsInterface` function.

Recommendation: Consider implementing the required ERC-165 `supportsInterface` function in the `GobblersERC1155B` contract.

ArtGobblers: Implemented in GobblersERC1155B.sol#L124 and PagesERC721.sol#L162.

Spearbit: Acknowledged.

5.3.2 `LogisticVRGDA` is importing `wadExp` from `SignedWadMath` but never uses it

Severity: *Informational*

Context: LogisticVRGDA.sol#L4

Description: The `LogisticVRGDA` is importing the `wadExp` function from the `SignedWadMath` library but is never used.

Recommendation: Remove the `wadExp` to improve readability.

```
-import {wadExp, wadLn, unsafeDiv, unsafeWadDiv} from "../lib/SignedWadMath.sol";
+import {wadLn, unsafeDiv, unsafeWadDiv} from "../lib/SignedWadMath.sol";
```

Art Gobblers: Recommendation implemented in the PR 111.

Spearbit: Acknowledged.

5.3.3 `Pages.tokenURI` does not revert when `pageId` is the ID of an invalid or not minted token

Severity: *Informational*

Context: Pages.sol#L207-L211

Description: The current implementation of `tokenURI` in `Pages` is returning an empty string if the `pageId` specified by the user's input has not been minted yet (`pageId > currentId`).

Additionally, the function does not correctly handle the case of a special `tokenId` equal to `0`, which is an invalid token ID given that the first mintable token would be the one with ID equal to `1`.

The EIP-721 documentation specifies that the contract should revert in this case:

> *Throws if `_tokenId` is not a valid NFT. URIs are defined in RFC 3986.*

Recommendation: When the specified `pageId` has not been minted yet or is invalid, make the `tokenURI` revert instead of returning an empty string.

ArtGobblers: Fixed in commit 3494802.

Spearbit: Acknowledged.

5.3.4 Consider checking if the token fed to the Gobbler is a real ERC1155 or ERC721 token

Severity: *Informational*

Context: ArtGobblers.sol#L672-L674

Description: The current implementation of `ArtGobblers.feedArt` function allows users to specify from the value of the `bool isERC1155` input parameter if the `id` passed is from an `ERC721` or `ERC1155` type of token.

Without checking if the passed `nft` address fully support `ERC721` or `ERC1155` these two problems could arise:

- The user can feed to a Gobbler an arbitrary ERC20 token by calling `gobblers.feedArt(1, address(goo), 100, false);`. In this example, we have fed 100 `$GOO` to the gobbler.
- By just implementing `safeTransferFrom` or `transferFrom` in a generic contract, the user can feed tokens that cannot later be rendered by a Dapp because they do not fully support `ERC721` or `ERC1155` standard.

Recommendation: Consider checking if the specified `nft` address passed by the user as an input parameter is a valid `ERC721` or `ERC1155` token.

5.3.5 Rounding down in legendary auction leads to `legendaryGobblerPrice` being zero earlier than the auction interval

Severity: *Informational*

Context: ArtGobblers.sol#L443

Description: The expression below rounds down.

```
startPrice * (LEGENDARY_AUCTION_INTERVAL - numMintedSinceStart)) / LEGENDARY_AUCTION_INTERVAL
```

In particular, this expression has a value 0 when `numMintedSinceStart` is between 573 and 581 (`LEGENDARY_-AUCTION_INTERVAL`).

Recommendation: Consider checking if this is intentional. If not, consider rounding up to the nearest integer so that the legendary gobbler price is only zero after the interval `LEGENDARY_AUCTION_INTERVAL`.

Art Gobblers: Fixed in dc7340a.

Spearbit: Acknowledged.

5.3.6 Typos in code comments or natspec comments

Severity: *Informational*

Context:

- Pages.sol#L179, Pages.sol#L188, Pages.sol#L205, LogisticVRGDA.sol#L23, VRGDA.sol#L34, ArtGobblers.sol#L54, ArtGobblers.sol#L745, ArtGobblers.sol#L754, ArtGobblers.sol#L606, ArtGobblers.sol#L871, ArtGobblers.sol#L421, ArtGobblers.sol#L435-L436, ArtGobblers.sol#L518, ArtGobblers.sol#L775 and ArtGobblers.sol#L781

Description: Below is a list of typos encountered in the code base and / or natspec comments:

- In both Pages.sol#L179 and Pages.sol#L188 replace `compromise` with `comprise`
- In Pages.sol#L205 replace `pages's URI` with `page's URI`
- In LogisticVRGDA.sol#L23 replace `effects` with `affects`
- In VRGDA.sol#L34 replace `actions` with `auctions`
- In ArtGobblers.sol#L54, ArtGobblers.sol#L745 and ArtGobblers.sol#L754 replace `compromise` with `comprise`
- In ArtGobblers.sol#L606 remove the double occurrence of the word `state`
- In ArtGobblers.sol#L871 replace `emission's` with `emission`
- In ArtGobblers.sol#L421 replace `gobblers is minted` with `gobblers are minted` and `until all legendaries been sold` with `until all legendaries have been sold`
- In ArtGobblers.sol#L435-L436 replace `gobblers where minted` with `gobblers were minted` and `if auction has not yet started` with `if the auction has not yet started`
- In ArtGobblers.sol#L518 replace `overflow we've got bigger problems` with `overflow, we've got bigger problems`
- In ArtGobblers.sol#L775 and ArtGobblers.sol#L781 replace `get emission emissionMultiple` with `get emissionMultiple`

Recommendation: We suggest fixing the abovementioned typos to improve legibility for the end user and other developers.

Art Gobblers: Recommendations implemented in the PR #111.

Spearbit: Acknowledged.

5.3.7 Missing natspec comments for contract's constructor, variables or functions

Severity: *Informational*

Description: Some of the contract's `constructor` variables and functions are missing natespec comments. Here is the full list of them:

- `Pages` `constructor`
- `Pages` `getTargetSaleDay` function
- `LibString` `toString` function
- `MerkleProofLib` `verify` function
- `SignedWadMath` `toWadUnsafe` function
- `SignedWadMath` `unsafeWadMul` function
- `SignedWadMath` `unsafeWadDiv` function
- `SignedWadMath` `wadMul` function
- `SignedWadMath` `wadDiv` function
- `SignedWadMath` `wadExp` function
- `SignedWadMath` `wadLn` function
- `SignedWadMath` `unsafeDiv` function
- `VRGDA` `constructor`
- `LogisticVRGDA` `constructor`
- `LogisticVRGDA` `getTargetDayForNextSale`
- `PostSwitchVRGDA` `constructor`
- `PostSwitchVRGDA` `getTargetDayForNextSale`
- `GobblerReserve` `artGobblers`
- `GobblerReserve` `constructor`
- `GobblersERC1155B` contract is missing natspec's coverage for most of the variables and functions
- `PagesERC721` contract is missing natspec's coverage for most of the variables and functions
- `PagesERC721` `isApprovedForAll` should explicity document the fact that the `ArtGobbler` contract is **always pre-approved**
- `ArtGobblers` `chainlinkKeyHash` variable
- `ArtGobblers` `chainlinkFee` variable
- `ArtGobblers` `constructor`
- `ArtGobblers` `gobblerPrice` miss the @return natspec
- `ArtGobblers` `legendaryGobblerPrice` miss the @return natspec
- `ArtGobblers` `requestRandomSeed` miss the @return natspec
- `ArtGobblers` `fulfillRandomness` miss both the @return and @param natspec
- `ArtGobblers` `uri` miss the @return natspec
- `ArtGobblers` `gooBalance` miss the @return natspec
- `ArtGobblers` `mintReservedGobblers` miss the @return natspec
- `ArtGobblers` `getGobblerEmissionMultiple` miss the @return natspec

- `ArtGobblers` `getUserEmissionMultiple` miss the @return natspec
- `ArtGobblers` `safeBatchTransferFrom` miss all natspec
- `ArtGobblers` `safeTransferFrom` miss all natspec
- `ArtGobblers` `transferUserEmissionMultiple` miss @notice natspec

Recommendation: Consider adding the missing natspec comments to the above listed items.

ArtGobblers: Fixed some of them in PR #111.

Spearbit: Acknowledged.

5.3.8 Potential issues due to slippage when minting legendary gobblers

Severity: *Informational*

Context: ArtGobblers.sol#L358-L360

Description: The price of a legendary mint is a function of the number of gobblers minted from goo. Because of the strict check that the price is exactly equal to the number of gobblers supplied, this can lead to slippage issues. That is, if there is a transaction that gets mined in the same block as a legendary mint, and before the call to `mintLegendaryGobbler`, the legendary mint will revert.

```
uint256 cost = legendaryGobblerPrice();
if (gobblerIds.length != cost) revert IncorrectGobblerAmount(cost);
```

Recommendation: Consider making the check less strict.

```
- if (gobblerIds.length != cost) revert IncorrectGobblerAmount(cost);
+ if (gobblerIds.length < cost) revert IncorrectGobblerAmount(cost);
```

Note: the loop that follows as well as the event also must be updated accordingly.

Art Gobblers: You could assume that anyone doing a legendary mint is probably using flashbots anyways, but we'd like this to be sound regardless. Fixed in PR #112.

Spearbit: Acknowledged.

5.3.9 Users who claim early have an advantage in goo production

Severity: *Informational*

Context: ArtGobblers.sol#L287

Description: The gobblers are revealed in ascending order of the index in `revealGobblers`. However, there can be cases when this favours users who were able to claim early:

1. There is the trivial case where a user who claimed a day earlier will have an advantage in `gooBalance` as their emission starts earlier.
2. For users who claimed the gobblers on the same day (in the same period between a reveal) the advantage depends on whether the gobblers are revealed in the same block or not.
    1. If there is a large number of gobbler claims between two aforementioned gobblers, then it may not be possible to call `revealGobblers`, due to block gas limit.
    2. A user at the beginning of the reveal queue may call `revealGobblers` for enough indices to reveal their gobbler early.

In all of the above cases, the advantage is being early to start the emission of the Goo.

Recommendation: We recommend the project to call `revealGobblers` with the largest possible number that can fit in the block. Similarly, we recommend calling `revealGobblers` as soon as `fulfillRandomness` is called.

5.3.10 Add a negativity check for `decayConstant` in the constructor

Severity: *Informational*

Context: VRGDA.sol#L26

Description: Price is designed to decay as time progresses. For this, it is important that the constant `decayConstant` is negative. Since the value is derived using an on-chain logarithm computation once, it is useful to check that the value is negative.

Also, typically decay constant is positive, for example, in radioactive decay the negative sign is explicitly added in the function. It is worth keeping the same convention here, i.e., keep `decayConstant` as a positive number and add the negative sign in `getPrice` function. However, this may cause a small increase in gas and therefore may not be worth implementing in the end.

Recommendation: Consider adding the following change.

```
  decayConstant = wadLn(1e18 - periodPriceDecrease);
+ assert(decayConstant < 0);
```

5.3.11 Improvements in `toString()`

Severity: *Informational*

Context: LibString.sol#L31, LibString.sol#L17 and LibString.sol#L10

Description:

1. The current `toString` function carefully calculates the offsets in such a way that the least significant bit of the offset will contain the 8-byte data corresponding to ASCII value of the number. Instead, we recommend simplifying the for loop by using `mstore8`.
2. There is a redundant `mstore(str, k)` at the beginning of the loop that can be removed.
3. Consider marking the assembly block as memory-safe and setting the version pragma to at least 0.8.13. This will allow the IR optimizer to lift stack variables to memory if a "stack-too-deep" error is encountered.

Art Gobblers: The suggestions (1) and (2) are implemented in LibString.sol from Solmate.

Spearbit: Acknowledged.

5.3.12 Consideration on possible Chainlink integration concerns

Severity: *Informational*

Context: ArtGobblers.sol#L485 and ArtGobblers.sol#L489-L496

Description: The ArtGobbler project relies on the Chainlink v1 VRF service to reveal minted gobblers and assign a random `emissionMultiple` that can range from 6 to 9.

The project has estimated that minting and revealing all gobblers will take about 10 years.

In the scenario simulated by the discussion "Test to mint and reveal all the gobblers" the number of `requestRandomSeed` and `fulfillRandomness` made to reveal all the minted gobblers were more than 1500.

Given the timespan of the project, the number of requests made to Chainlink to request a random number and the fundamental dependency that Chainlink VRF v1 has, we would like to highlight some concerns:

- What would happen if Chainlink completely discontinues the Chainlink VRF v1? At the current moment, Chainlink has already released VRF v2 that replaces and enhances VRF v1.
- What would happen in case of a Chainlink service outage and for some reason they decide not to process previous requests? Currently, the `ArtGobbler` contract does not allow to request a new "request for randomness".

- What if the `fulfillRandomness` always gets delayed by a long number of days and users are not able to reveal their gobblers? This would not allow them to know the value of the gobbler (rarity and the visual representation) and start compounding `$GOO` given the fact that the gobbler does not have an emission multiple associated yet.
- What if for error or on purpose (malicious behavior) a Chainlink operator calls `fulfillRandomness` multiple times changing the `randomSeed` during a reveal phase (the reveal of X gobbler can happen in multiple stages)?

Recommendation: Given the important dependency that Chainlink VRF v1 has for the project, we suggest evaluating the scenarios listed above and verify with the Chainlink team what are the possible security countermeasures to take.

ArtGobblers: Fixed in PR #114.

Spearbit: Acknowledged.

5.3.13 The function `toString()` does not return a string aligned to a 32-byte word boundary

Severity: *Informational*

Context: LibString.sol#L38

Description: It is a good practice to align memory regions to 32-byte word boundaries. This is not necessarily the case here. However, we do not think this can lead to issues.

Recommendation: Consider changing the string conversion algorithm so that the starting string (and correspondingly, the end) will be at 32-byte word boundaries.

5.3.14 Considerations on Legendary Gobbler price mechanics

Severity: *Informational*

Context: ArtGobblers.sol#L408 and ArtGobblers.sol#L418-L445

Description: The auction price model is made in a way that starts from a `startPrice` and decays over time. Each time a new action starts the price time will be equal to `max(69, prevStartPrice * 2)`.

Users in this case are incentivized to buy the legendary gobbler as soon as the auction starts because by doing so they are going to burn the maximum amount allowed of gobblers, allowing them to maximize the final emission multiple of the minted legendary gobbler.

By doing this, you reach the end goal of maximizing the account's `$GOO` emissions. By waiting, the cost price of the legendary gobbler decays, and it also decays the emission multiple (because you can burn fewer gobblers).

This means that if a user has enough gobblers to burn, he/she will burn them as soon as the auction starts.

Another reason to mint a legendary gobbler as soon as the auction starts (and so burn as many gobblers as possible) is to make the next auction starting price as high as possible (always for the same reason, to be able to maximize the legendary gobbler emissions multiple).

The next auction starting price is determined by `legendaryGobblerAuctionData.startPrice = uint120(cost < 35 ? 69 : cost << 1);`

These mechanisms and behaviors can result in the following consequences:

- Users that will have a huge number of gobblers will burn them as soon as possible, disallowing others that can't afford it to wait for the price to decay.
- There will be less and less "normal" gobblers available to be used as part of the "art" aspect of the project. In the discussion "Test to mint and reveal all the gobblers" we have simulated a scenario in which a whale would be interested to collect all gobblers with the end goal of maximizing `$GOO` production. In that scenario, when the last Legendary Gobbler is minted we have estimated that 9644 gobbler have been burned to mint all the legendaries.

Recommendation: If the following scenarios don't fit your expected behavior consider tweaking the current legendary gobbler price mechanics to avoid them.

- Most of the gobblers will be burned to mint the legendary gobblers.
- Only users that own a large number of gobblers can afford to mint legendary gobblers.

5.3.15 Define a `LEGENDARY_GOBBLER_INITIAL_START_PRICE` constant to be used instead of hardcoded 69

Severity: *Informational*

Context: ArtGobblers.sol#L274 and ArtGobblers.sol#L408

Description: `69` is currently the starting price of the first legendary auction and will also be the price of the next auction if the previous one (that just finished) was lower than `35`.

There isn't any gas benefit to use a `constant` variable but it would make the code cleaner and easier to read instead of having hard-coded values directly.

Recommendation: Consider creating a `uint256 public constant LEGENDARY_GOBBLER_INITIAL_START_PRICE = 69;` and replace it where the value `69` is currently hard-coded inside the code.

ArtGobblers: Fixed in commit a51fca1.

Spearbit: Acknowledged.

5.3.16 Update `ArtGobblers` comments about some variable/functions to make them more clear

Severity: *Informational*

Context: ArtGobblers.sol#L123, ArtGobblers.sol#L169, ArtGobblers.sol#L397 and ArtGobblers.sol#L435-L437

Description: Some comments about state variables or functions could be improved to make them clearer or remove any further doubts.

`LEGENDARY_AUCTION_INTERVAL`

> /// @notice Legendary auctions begin each time a multiple of these many gobblers have been minted.

It could make sense that this comment specifies "minted from Goo" otherwise someone could think that also the "free" mints (mintlist, legendary, reserved) could count to determine when a legendary auction start.

`EmissionData.lastTimestamp`

> // Timestamp of last deposit or withdrawal.

These comments should be updated to cover all the scenarios where `lastBalance` and `lastTimestamp` are updated. Currently, they are updated in many more cases for example:

- `mintLegendaryGobbler`
- `revealGobblers`
- `transferUserEmissionMultiple`

`getGobblerData[gobblerId].emissionMultiple = uint48(burnedMultipleTotal << 1)` has an outdated comment.

The current line `getGobblerData[gobblerId].emissionMultiple = uint48(burnedMultipleTotal << 1)` present in the `mintLegendaryGobbler` function has the following comment:

> // Must be done before minting as the transfer hook will update the user's emissionMultiple.

In both `ArtGobblers` and `GobblersERC1155B` there isn't any transfer hook, which could mean that the referred comment is referencing outdated code. We suggest removing or updating the comment to reflect the current code implementation.

`legendaryGobblerPrice` `numMintedAtStart` calculation.

The variable `numMintedAtStart` is calculated as `(numSold + 1) * LEGENDARY_AUCTION_INTERVAL` The comment above the formula does not explain why it uses `(numSold + 1)` instead of `numSold`. This reason is correctly explained by a comment on `LEGENDARY_AUCTION_INTERVAL` declaration.

It would be better to also update the comment related to the calculation of `numMintedAtStart` to explain why the current formula use `(numSold + 1)` instead of just `numSold`

`transferUserEmissionMultiple`

The above utility function transfers an amount of a user's emission's multiple to another user. Other than transferring that emission amount, it also updates both users `lastBalance` and `lastTimestamp`

The `natspec` comment should be updated to cover this information.

Recommendation: Consider updating the referenced comments or removing the outdated ones to improve clarity.

ArtGobblers: Fixed in PR #111.

Spearbit: Acknowledged.

5.3.17 Mark functions not called internally as `external` to improve code quality

Severity: *Informational*

Context: Goo.sol#L51, Goo.sol#L58, Goo.sol#L65 and GobblerReserve.sol#L30

Description: The following functions could be declared as `external` to save gas and improve code quality:

- `Goo.mintForGobblers`
- `Goo.burnForGobblers`
- `Goo.burnForPages`
- `GobblerReserve.withdraw`

Recommendation: Consider declaring those functions as `external` to save gas and improve code quality.

Art Gobblers: Recommendation implemented in PR 111.

Spearbit: Acknowledged.

6 Appendix

6.1 Goo production when pooling gobblers

The goo emitted (Δs) for duration Δt is given by

$$\Delta s = \frac{1}{4}\left(m\Delta t^2\right) + \sqrt{s \cdot m}\Delta t$$

Assume there are two gobblers, 1 and 2, with corresponding goo being s_1 and s_2. Consider the strategy of combining the two gobblers into a single account. We want to see if this strategy improves goo production as opposed to using them separately.

$$\Delta s_1 = \frac{1}{4}\left(m_1\Delta t^2\right) + \sqrt{s_1 \cdot m_1}\Delta t$$

$$\Delta s_2 = \frac{1}{4}\left(m_2\Delta t^2\right) + \sqrt{s_2 \cdot m_2}\Delta t$$

$$\Delta s = \frac{1}{4}\left((m_1 + m_2)\,\Delta t^2\right) + \sqrt{(s_1 + s_2) \cdot (m_1 + m_2)}\Delta t$$

The difference is the following

$$\Delta s - \Delta s_1 - \Delta s_2 = \left(\sqrt{(s_1 + s_2) \cdot (m_1 + m_2)} - \sqrt{s_1 \cdot m_1} - \sqrt{s_2 \cdot m_2}\right) \cdot \Delta t \tag{1}$$

The equation (1) is always non-zero. To see this, use Cauchy-Schwarz $|x^2 + y^2| \cdot |a^2 + b^2| \geq |ax + by|^2$ where vectors - $(x, y) = (\sqrt{s_1}, \sqrt{s_2})$ - $(a, b) = (\sqrt{m_1}, \sqrt{m_2})$

The pooling strategy is always better, except when equality holds for Cauchy-Schwarz: this happens when

$$\frac{s_1}{m_1} = \frac{s_2}{m_2}$$

The above argument can be generalized to n gobblers.

In short, pooling gobblers optimizes Goo production. If this was false, users with multiple gobblers would be incentivized to split it across multiple accounts, leading to what's arguably a bad game dynamic. The above results also hint at strategies to replicate the pooled portfolio using an unpooled portfolio of gobblers.

6.1.1 Formal proof written in SMTLIB and verified using Z3

During the review, we wrote a formal proof of the above that can be proven in Z3. This was unstreamed to the repo and can be found in `goo_pooling.smt`.

6.1.2 Replicating a pooled portfolio of Gobblers and Goo with an unpooled portfolio

Given n accounts $1, \cdots n$ each having a single gobbler with multiple m_i and a gobbler balance of s_i. Assume that there is a total of G Goo that's available to be distributed among each accounts: we want to compute the optimal allocation of Goo. Let us say that each account i receives s'_i amount of Goo, i.e., $\sum_{i=1}^{n} s'_i = G$.

From the above equation 1, we see that this is equivalent to maximizing the following expression:

$$\sum_{i=1}^{n} \sqrt{\left(s_i + s'_i\right) \cdot m_i}$$

From Cauchy-Schwarz, the sum is maximized when

$$\frac{s_1 + s'_1}{m_2} = \frac{s_2 + s'_2}{m_2} = \cdots = \frac{s_n + s'_n}{m_n} = t$$

From this, we can see that

$$t = \frac{\sum_{i=1}^{n} s_i + G}{\sum_{i=1}^{n} m_i}$$

And that the share s'_i is given by

$$s'_i = \frac{m_i}{\sum_{i=1}^{n} m_i} \left(\sum_{i=1}^{n} s_i + G \right) - s_i \tag{2}$$

The value s'_i can be negative, which corresponds to taking out Goo from the gobbler! That is, there are cases where the optimal goo production is

A special case of (2) happens when $G = 0$. This corresponds to the case where a group redistributes Goo to maximize the total Goo production. The equation (2), in this case can be formulated as:

$$s'_i + s_i = \frac{m_i}{\sum_{i=1}^{n} m_i} \left(\sum_{i=1}^{n} s_i \right)$$

That is, everyone's Goo amount is proportional to the multiple of the Gobbler.

We emphasize again that this same optimum can be reached by pooling all Gobblers and Goo in a single account, without having to go through the redistribution. Also note that once the redistribution happens, it continues being optimal as time progresses, which also simplifies the game mechanics.

6.2 Fuzzing the functions `wadln` and `wadexp`

Solmate's functions `lnWad` and `expWad` were used in ArtGobblers. These functions were implemented by Remco.

Since, this was one of the earliest applications of these functions, we built two different fuzzers to compare the implementation against a multi-precision implementation of exponentiation and logarithm.

A fast implementation in rust can be found in solmate-math-fuzz. This implementation compares the results of random inputs for `ln` and `exp` against a multi precision floating point library in Rust. A relative tolerance was chosen as these are numerical approximations. Also, an implementation that uses foundry and ffi against python was also done.

6.3 The Fisher-Yates shuffle

The Gobbler's multipliers are revealed using an on-chain Fisher-Yates shuffle. We wanted to empirically verify how random these shuffles looked.

We wrote some simple simulations for revealing all gobblers and visualized the shuffles in a jupyter notebook shuffles.ipynb. The multipliers 6, 7, 8 and 9 should occur with frequencies multiple 3054, 2618, 2291 and 2027. These numbers where chosen so that the 'weighted multiplier' for each of the group is approximately the same, i.e., $6 \times 3054 \approx 7 \times 2618 \approx 8 \times 2291 \approx 9 \times 2027$.

Here are two example plots of the distribution of multipliers:

6.4 Empirical analysis of Fixed points in the shuffle

It is interesting to look at the number of fixed points in the shuffle and compare them against the expected value of fixed points (i.e., shuffles that keep one of the indices unchained) in a random shuffle. It is known that for a set of n elements, the number of disarrangements (i.e., a shuffle that doesn't have any fixed points) converges to $\frac{1}{e}$ (reference). We found that the empirical data matched with this value. Details can be found in shuffles.ipynb.

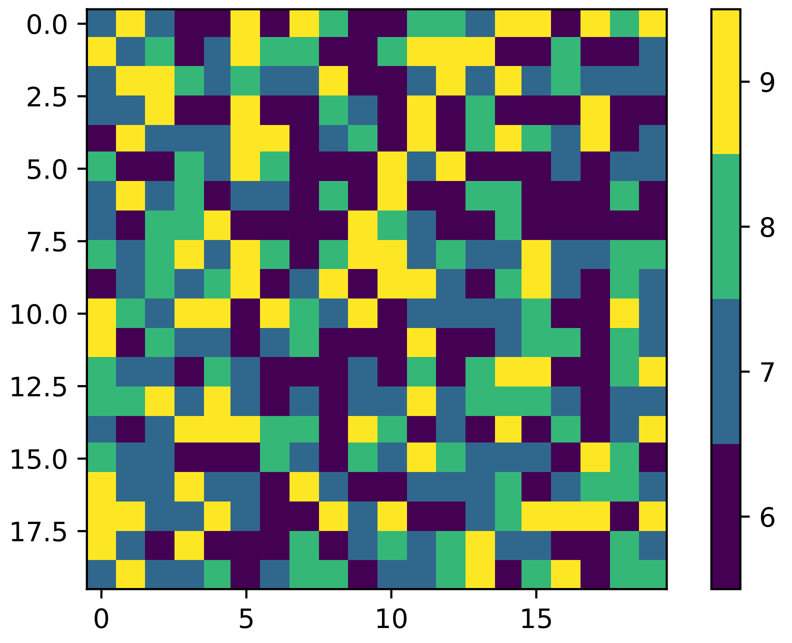

Figure 1: A sample from the distribution of multipliers

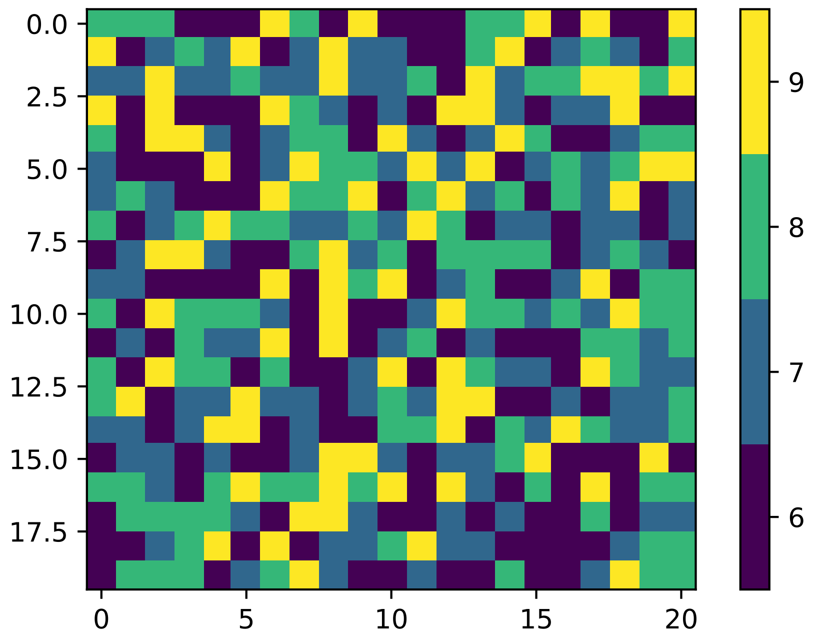

Figure 2: Another sample from the distribution of the multipliers

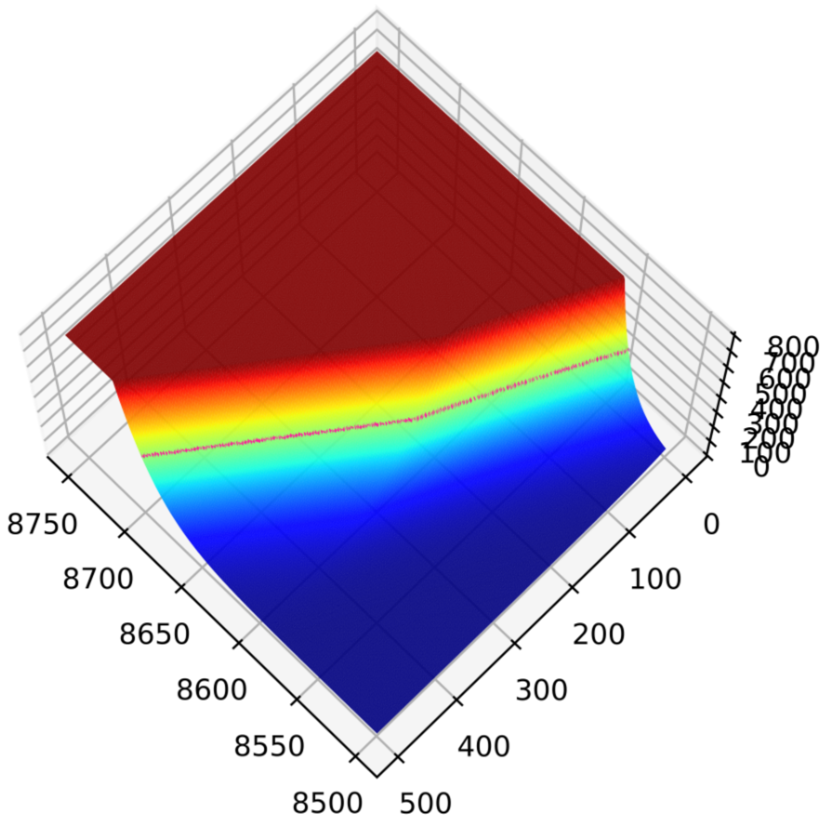

Figure 3: 3D Plot of Page VRGDA Price development, zoomed in at the time of the switch

6.5 The switch in Pages's VRGDA

The issuance of both Gobblers and Pages is based on VRGDAs, however, there's a limited amount of Gobblers that can be minted while the supply of Pages is infinite. Similarly to Gobblers the issuance schedule is fast at the beginning and then intended to slow down. But for Pages a switch happens at a certain point where issuance becomes linear.

One of the concerns here was whether this switch will be smooth, and to determine this, various plots were generated from the contract's implementation of the VRGDA logic. One can plot this in three-dimensional space: The time that has passed, the amount of Pages minted so far and the price resulting from these two parameters.

Zoomed in at the point of the switch, there appeared to be a slight dent. To further investigate this, a two-dimensional chart was plotted with the price fixed at 4.20, which is the target starting price of each Page's auction.

The plot shows how much time has to pass between each sale to reach the intended starting price again. Here the issue became clearly visible: There's a sudden drop of time during the switch to linear issuance.

The smoothness-problem was acknowledged and fixed by changing the Page's VRGDA parameters.

6.6 Note on solutions of the differential equation

The rate of GOO issuance, $g(t) : [0, \infty) \to [0, \infty)$ is defined by the boundary value problem:

$$g'(t) = \sqrt{m \cdot g(t)}$$

where $g(0) = c$ for some constant c, the initial value of GOO.

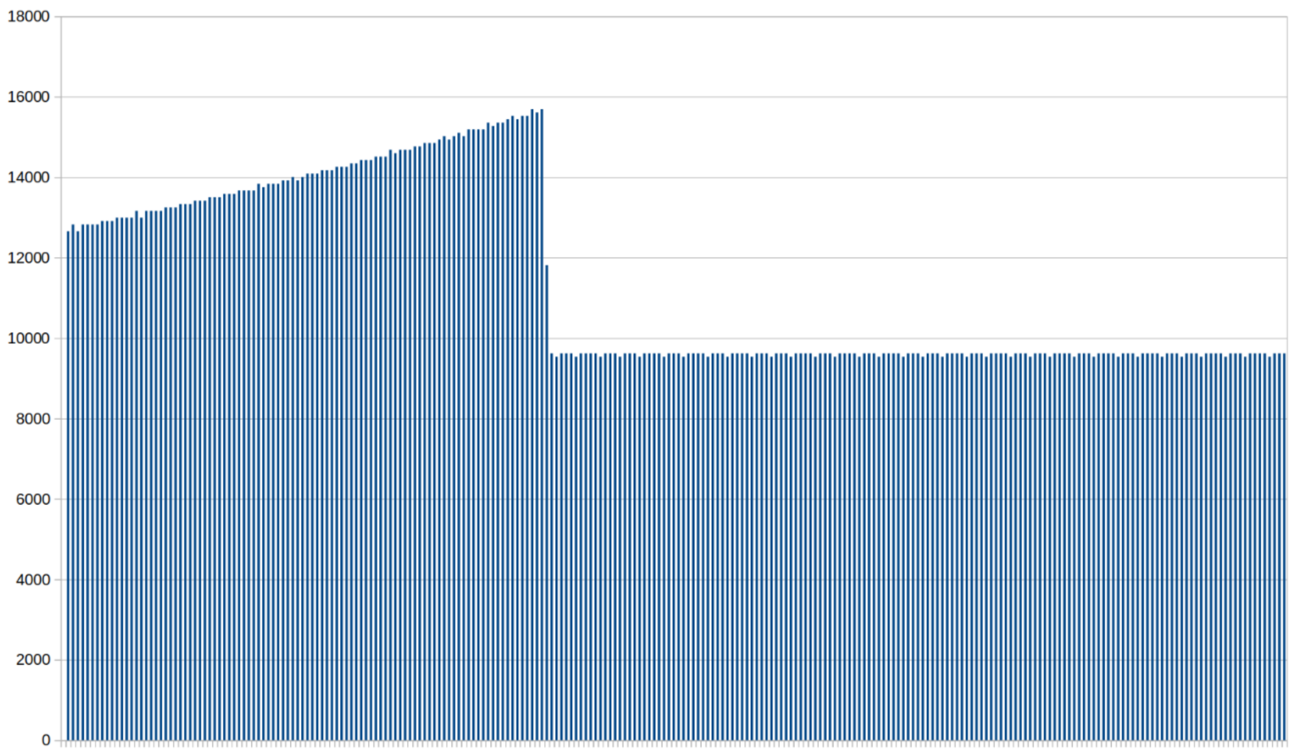

Figure 4: 2D Plot of time passed until Page price reaches the target again, zoomed in at the time of the switch

One of the solutions for the differential equation is

$$g(t) = \frac{1}{4}\left(\sqrt{m}t + 2\sqrt{c}\right)^2$$

However, for $c = 0$, the above solution is not unique, as $g(t) = 0$ is also another solution.

One can notice that the function $\sqrt{m \cdot y}$ is not Lipschitz continuous at $y = 0$. To see this, assume that there exists a $K > 0$ such that for all distinct $y_0, y_1 \in \mathbb{R}$ (without loss of generality, let's assume that they are also non-negative):

$$\begin{aligned} |\sqrt{m \cdot y_0} - \sqrt{m \cdot y_1}| \leq K \cdot |y_0 - y_1| \Rightarrow \\ \frac{\sqrt{m}}{\sqrt{y_0} + \sqrt{y_1}} \leq K \end{aligned} \tag{3}$$

However, for any two distinct Reals y_0, y_1 satisfying $\sqrt{y} < \frac{\sqrt{m}}{2K}$, the inequality (3) is false.

If the range of g is in (ε, ∞) for some $\varepsilon > 0$, it is straightforward to see that $\sqrt{m \cdot g}$ is Lipschitz continuous, which means that there is a unique solution for the differential equation for such cases due to Picard–Lindelöf theorem.

6.7 About legendary gobbler mints without revealing gobblers

Apart from the issues mentioned in findings, Spearbit left a comment in the shared GitHub repo that `mintLegendaryGobbler` allows using un-revealed gobblers to mint legendary gobblers, and uses the multiplier value of 0 for such cases. This may not be ideal in some cases.

The art-gobblers team commented that (paraphrasing):

> This is great thinking but I think we're probably ok with the current functionality. Like if there's an arms race for legendary gobblers we might not want ppl to have to wait until 4:20 to have their gobblers revealed before they can use them to snatch up the legendary. But, I'll check with others in team just to be sure.

Yeah, the team is also on the side of keeping as is.

6.8 Changing Gobbler NFT from ERC-1155 to ERC-721

After the Spearbit review, the Gobbler NFT was changed from ERC-1155 to ERC-721 (see PR #144). This introduced a very subtle bug: the approvals in ERC-1155 standard (`isApprovedForAll`) indexes on the owner of the NFT, whereas, approvals in ERC-721 indexes on the NFT `id`. This means that, a destructive action on the NFT, such as burning the gobblers to mint a legendary will need to remove this approval. However, this was missed in the refactor, which allows these approved and burned tokens to be resurrected, breaking the protocol. This bug wasn't present in original ERC-1155 implementation due to the subtle difference in how approvals are made.