yAudit GuessOurBlock Review

Review Resources:

- code repository

Auditors:

- Panda
- HHK

Table of Contents

Review Summary

GuessOurBlock

GuessOurBlock is a game in which users can guess the block number that will be made by a heroglyph validator. The user who guesses the winning block number will receive rewards.

The contracts of the GuessOurBlock Repo were reviewed over four days. Two auditors performed the code review between 24 September and 27 September 2024. The repository was under active development during the review, but the review was limited to the latest commit for the GuessOurBlock repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src
├── GuessOurBlockReceiver.sol
├── GuessOurBlockSender.sol
├── IGuessOurBlock.sol
└── dripVaults
    ├── BaseDripVault.sol
    ├── IDripVault.sol
    └── implementations
        ├── AaveVault.sol
        ├── MockVault.sol
        └── apxETHVault.sol
```

After the findings were presented to the GuessOurBlock team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, GuessOurBlock and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The contracts implement appropriate access control mechanisms with owner-only functions for sensitive operations.

Category	Mark	Description
Mathematics	Good	The mathematical operations are simple and appear to be implemented correctly, with proper use of precision and constants.
Complexity	Good	The codebase is straightforward to understand, with clear function purposes.
Libraries	Good	The project uses standard libraries like OpenZeppelin.
Decentralization	Average	The system allows community voting to disable certain features, promoting decentralization. However, some centralized control remains with the owner.
Code stability	Good	The code appears stable, with well-defined functions and structures. However, the ability to change critical parameters like `groupSize` could lead to instability.
Documentation	Average	The contracts are partially documented.
Monitoring	Average	Some state variables are not logged.
Testing and verification	Average	Most key functions are tested, but more comprehensive testing could be beneficial, especially for edge cases like updating `groupSize`.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational

- Findings including recommendations and best practices.

Critical Findings

None.

High Findings

None.

Medium Findings

1. Medium - Potential for rug pull via `setGob` function

During our discussion with the team, it was mentioned that the `updateDripVault()` function can be invoked by the team to transfer all funds to the team's treasury address. If the community disapproves of this mechanism, they have the ability to vote and disable this feature.

However, even with the feature disabled, the team retains the ability to withdraw all funds from the drip vaults.

Technical Details

The `setGob()` function within the BaseDripVault contract allows the contract owner to modify the gob address. Suppose the owner is compromised or acts maliciously. In that case, they can set the gob address to one they control, enabling them to withdraw all funds from the vault regardless of whether `updateDripVault()` is enabled or disabled.

Impact

Medium. The team can still take the user funds even when `updateDripVault()` is disabled.

Recommendation

If possible, remove the `setGob()` function.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/1f89e4d7a6fe5da48f6c08398fd9dabd5b96a99c.

2. Medium - Updating `groupSize` can make existing guesses unwinnable

The current implementation of the `updateGroupSize` function can make existing valid guesses unwinnable after changing the `groupSize`, whether increasing or decreasing it. This occurs because the `_blockTailNumber` calculation changes, potentially misaligning existing guesses with their intended blocks.

Technical Details

1. Users make guesses based on the current `groupSize`.
2. The `_blockTailNumber` is calculated as `blockNumber - (blockNumber % groupSize)`.
3. If `groupSize` is changed (increased or decreased), the `_blockTailNumber` for existing guesses may change.
4. When processing winners, the code checks against the new `_blockTailNumber`, potentially missing valid guesses.

** Example:**

1. Decreasing `groupSize`:
   - Initial `groupSize`: 10
   - User guesses for block 101, resulting in `_blockTailNumber` 100
   - `groupSize` is updated to 3

- New `_blockTailNumber` for block 101 becomes 99

2. Increasing `groupSize`:
   - Initial `groupSize`: 10
   - User guesses for block 101, resulting in `_blockTailNumber` 100
   - `groupSize` is updated to 11
   - New `_blockTailNumber` for block 101 becomes 99

If block 101 wins, the code checks winners for `_blockTailNumber` 99, missing the user's guess.

Impact

Medium. Users who made valid guesses under the previous `groupSize` may lose their chance to win if the `groupSize` changes in either direction, leading to potential loss of funds.

Recommendation

Remove `updateGroupSize()`, set `groupSize` via the constructor, and deploy a new contract if it's to be changed.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/b44aa510315cddcb9c588b5f67a0c9997198948c.

3. Medium - Changing `groupSize` can make previously losing guesses become winners

Modifying the `groupSize` in the GuessOurBlockReceiver contract can cause previously losing guesses to become winners, leading to potential conflicts and unfair distribution of rewards. Additionally, this change invalidates the `totalGuessWeight` for affected blocks, further complicating the reward distribution process.

Technical Details

1. Users make guesses based on the current `groupSize`.
2. The `_blockTailNumber` is calculated as `blockNumber - (blockNumber % groupSize)`.
3. If `groupSize` is changed, some blocks that were previously not tail blocks may become tail blocks.
4. This can result in guesses that were previously losing becoming winning guesses.

5 The `totalGuessWeight` for each block is now incorrect, as it doesn't account for the newly included or excluded guesses.

** Example:**

- Initial `groupSize`: 10
- User A guesses for block 102, resulting in `_blockTailNumber` 100
- User B guesses for block 112, resulting in `_blockTailNumber` 110
- Block 105 is the actual winning block so that User A would win
- `totalGuessWeight` for block 100 is calculated based on these guesses
- `groupSize` is updated to 20
- New `_blockTailNumber` for both blocks 102 and 112 becomes 100
- Now both User A and User B have winning guesses for the same block
- The `totalGuessWeight` for block 100 is incorrect, as it doesn't include weights from guesses for blocks 110-119.

Impact

Medium. The change in `groupSize` disrupts the fairness and integrity of the reward distribution system.

Recommendation

Disable `groupSize` change, add it as a constructor parameter, and deploy a new contract if it's to be changed.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/b44aa510315cddcb9c588b5f67a0c9997198948c.

4. Medium - `onValidatorTriggered()` should be `payable`

The function `onValidatorTriggered()` integrates with LayerZero however it is non-payable which is an issue as the integration require a `msg.value` to pay the gas fees.

Technical Details

The function `onValidatorTriggered()` is supposed to be called by the relayer whenever a Heroglyph validator created a block.

The function doesn't have the `payable` keywords, which results in Solidity not allowing a `msg.value` different than 0 when the function is called. This is an issue as this function integrates LayerZero and calls `_quote()` with `_payInLzToken` set to `false` before calling `_lzSend()`. This will require a `msg.value` different than 0 to pay for the gas on the destination chain. Because of this, validators won't be able to draw winners.

POC:

```
// SPDX-License-Identifier: Unlicense
pragma solidity >=0.8.0;

import "forge-std/Test.sol";
import { GuessOurBlockSender } from "src/GuessOurBlockSender.sol";
import "@layerzerolabs/lz-evm-oapp-v2/contracts/oapp/OAppSender.sol";

contract FakeEndpoint {
    function setDelegate(address) external {
        return;
    }

    function quote(MessagingParams calldata _params, address _sender) external view
returns (MessagingFee memory) {
        return MessagingFee({
            nativeFee: 100 wei, //pay in lz is set to false so return amount only for
native
            lzTokenFee: 0 wei
        });
    }
}

contract POC is Test {
    function test_notPayable() public {
        FakeEndpoint fake = new FakeEndpoint();
        GuessOurBlockSender gob = new GuessOurBlockSender(1, address(fake),
address(this), address(address(this)));
        vm.deal(address(gob), 100 ether); //give eth to gob

        gob.setPeer(1, bytes32(uint256(1)));

        //try with no value
        vm.expectRevert(abi.encodeWithSelector(OAppSender.NotEnoughNative.selector, 0));
        gob.onValidatorTriggered(0, 10, address(gob), 0);
```

```
        //try low level call with value
        (bool success,) = payable(gob).call{value: 100 wei}
(abi.encodeCall(gob.onValidatorTriggered, (0, 10, address(gob), 0)));
        assertEq(success, false);
    }
}
```

Impact

Medium.

Recommendation

Make the function payable or pay the gas using LZ tokens.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/commit/ed49743962dbb12c1a4be8d1161d4550bc102cbc.

5. Medium - Only past bets should be winnable

Users can only place bets for blocks within 24 hours of the current time. However, if someone wins during the 24 hours, they will receive the money from all bets, including the one that was just placed for a future block. This results in high risk and very low rewards for users.

Technical Details

When a user wants to place a bet, they can call the functions `guess()` or `multiGuess()`. They can only select a block that will be created 24 hours or later from now.

However, during these two functions, the amount bet by the user is added to the `lot`, which is the the stored sum of all bets. When a Heroglyph validator creates a block the function `_lzReceive()` during which the `lot` will be distributed minus some fees to the winning bets.

This means the only way to profit as a new user is to either hope no one gets the block right in the next 24 hours or that if someone does, they either didn't bet enough to have full weight, and then the distributed reward is reduced, or more people bet after, so the `lot` increases. 15% is automatically replenished for the next draw inside `_lzReceive()` but that's only if it's greater than TOO_LOW_BALANCE and even in that case it might end up being less than what the user bet.

Impact

Medium.

Recommendation

- Consider only distributing the money from past and current bets but not from bets in the future.
- Consider setting a minimum amount per bet to limit spam.

Developer Response

Partially fixed in https://github.com/HeroglyphEVM/GuessOurBlock/commit/73bcb5bde8fe158f48291b053c3ea00f7700259c.

New bets can still be won before they are active but the minimum duration until a bet can be active has been reduced from 1 day to 33 blocks.

6. Medium - `Validator` can block `lzReceive()`

The `GuessOurBlockReceiver` contract integrates with LayerZero, the function in charge of receiving messages makes external call to the `validator` which might want to revert on purpose for malicious reasons.

Technical Details

When Heroglyph creates a block, a LayerZero message is sent with the `validator` and the `blockNumber` found in the `GuessOurBlockReceiver` contract.

The `_lzReceive()` function will receive that message, compute the rewards and then the fees in ETH to the `validator` and `treasury`.

If the `validator` is a smart contract, it could revert on purpose or use more gas than was passed by the original LayerZero transaction when receiving the fee transfer. Later, the validator could retry the LayerZero message when conditions are right.

One case for this example would be if the `validator` wanted to wait for the `lot` to be bigger so a bigger share could be earned. The validator could revert as long as the `lot` is not big enough, which will block the protocol and rewards for the users until a new non-malicious `validator` creates a block with a bet on or that the `lot` is big enough so the malicious validator stops reverting on purpose.

Another case would be if the `validator` is just malicious and wants to deny rewards from block winners. In that case, users who bet for that block will never receive their rewards.

Impact

Medium.

Recommendation

Make a `claim()` function for the validator so he is not called during `lzReceive()` or wrap for `weth` prior to the transfer.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/af8a9d88aaa429646a259bc31d49fd9bd7a8e124.

Low Findings

1. Low - Voting incentive Imbalance near `_minimumBlockAgeInBlock`

The current voting mechanism incentivizes users to guess at the edge of `_minimumBlockAgeInBlock`, potentially leading to suboptimal voting patterns and unfair deposit loss.

Technical Details

The current voting mechanism allows users to guess for blocks that are _minimumBlockAgeInBlock or older. However, this creates an imbalance in the voting incentives:

1. Users who guess for blocks close to the _minimumBlockAgeInBlock have a higher chance of recovering their investment and potentially earning rewards.
2. Users who guess for blocks further in the future face a higher risk of losing their deposit, even if their chosen block is ultimately a winning one.

This is because if a winner is declared before a user's chosen block is reached, that user's deposit may be lost, regardless of the validity of their guess.

Impact

Low.

Recommendation

- Do not allow a guess to be placed too long in the future.
- Only distribute rewards up to the guesses made at the winning block

Developer Response

Acknowledged.

2. Low - `updateLzGasLimit()` doesn't update `defaultLzOption`

Technical Details

The function `updateLzGasLimit()` updates the variable `lzGasLimit` however this variable is only used in the constructor to build the variable `defaultLzOption` that will be used in `_lzSend()`.

This makes this setter (and variable) useless. In the case of the initial gas limit not being high enough or being too high, the owner won't be able to update it and will have to deploy a new contract.

Impact

Low.

Recommendation

When changing the ' lzGasLimit ', rebuild the variable `defaultLzOption`. You could consider removing that variable and directly updating the `defaultLzOption`.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/7f2f3e8faba2388c5464f4b868b9bf597c4594d0.

3. Low - Potential reentrancy in Drip vault's `withdraw()`

Technical Details

There is a reentrancy opportunity in the `withdraw()` function because it first calls the `_beforeWithdrawal()` internal function before updating its storage. This internal function will be making external call for `apxETHVault` and `AaveVault` that may create some reentrancy opportunity. In The `AaveVault`, the `rateReceiver` and `_to` addresses will be called as part of an ETH transfer; these addresses can reenter the protocol before `totalDeposit` is updated for the vaults.

While it currently doesn't result in any meaningful exploit, it is important to ensure that contracts cannot be reentered before the storage is updated.

Impact

Low.

Recommendation

Consider applying one or both of these suggestions:

- Update the storage variables before making external calls, especially when transferring ETH with `address.call()`.
- Use nonreentrant modifiers.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/commit/be4d9907fa3f9b0ca96349171255ad0c33109115 and https://github.com/HeroglyphEVM/GuessOurBlock/commit/d58d7963256038486485d93d747d6bc4a39bfca5.

Gas Saving Findings

1. Gas - State variables only set in the constructor should be declared `immutable`

Immutable variables save gas.

Technical Details

```
File: src/dripVaults/implementations/AaveVault.sol

29: aaveV3Pool = IAaveV3Pool(_aaveV3Pool);

30: weth = IWETH(_weth);
```

src/dripVaults/implementations/AaveVault.sol#L29,
src/dripVaults/implementations/AaveVault.sol#L30

```
File: src/dripVaults/implementations/apxETHVault.sol

27: apxETH = IApxETH(_apxETH);

28: pirexEth = IPirexEth(apxETH.pirexEth());
```

src/dripVaults/implementations/apxETHVault.sol#L27,
src/dripVaults/implementations/apxETHVault.sol#L28

Impact

Gas savings.

Recommendation

Use immutable variables.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/f3ba0b498b67177a432f5a82fe0f2ece2b12e427.

2. Gas - State variables can be packed into fewer storage slots

If variables occupying the same slot are both written using the same function or by the constructor, a separate Gsset (20000 gas) is avoided. Reads of the variables can also be cheaper.

Technical Details

File: src/GuessOurBlockReceiver.sol

```
// @audit: 2 slots could be saved, by using a different order:
\*
 * struct IGuessOurBlock.FeeStructure feeBps;
 * uint256 PRECISION; // (256 bits)
 * mapping(uint32 => struct IGuessOurBlock.BlockMetadata) blockData; // (256 bits)
 * mapping(address => mapping(uint32 => struct IGuessOurBlock.BlockAction)) actions; //
(256 bits)
 * address treasury; // (160 bits)
 * uint32 MAX_BPS; // (32 bits)
 * uint32 ONE_DAY_IN_ETH_BLOCK; // (32 bits)
 * uint32 groupSize; // (32 bits)
 * contract IDripVault dripVault; // (160 bits)
 * uint32 minimumBlockAge; // (32 bits)
 * bool isMigratingDripVault; // (8 bits)
 * bool permanentlySetDripVault; // (8 bits)
 * uint128 TOO_LOW_BALANCE; // (128 bits)
 * uint128 fullWeightCost; // (128 bits)
 * uint128 lot; // (128 bits)
 */

14: uint256 private constant PRECISION = 1e18;
15:     uint32 public constant MAX_BPS = 10_000;
16:     uint128 public constant TOO_LOW_BALANCE = 0.1e18;
17:     uint32 public constant ONE_DAY_IN_ETH_BLOCK = 7200;
18:
19:     FeeStructure private feeBps;
20:     address public treasury;
21:     IDripVault public dripVault;
22:
23:     // 1 complete ticket cost
24:     uint128 public fullWeightCost;
25:     uint128 public lot;
```

```
26:     uint32 public groupSize;
27:
28:     mapping(uint32 blockId => BlockMetadata) private blockData;
29:     mapping(address user => mapping(uint32 blockId => BlockAction)) private actions;
30:
31:     uint32 public minimumBlockAge;
32:
33:     bool public isMigratingDripVault;
34:     bool public permanentlySetDripVault
```

src/GuessOurBlockReceiver.sol#L14

Impact

Gas savings

Recommendation

Re-order state variables.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/3eb62ff25613432b095d9312384e17ce5baf68ce.

3. Gas - Refactor `AaveVault` `_beforeWithdrawal()`

The `_beforeWithdrawal()` function withdraws all funds from Aave to assess the yield generated. Yield can also be evaluated using the `balanceOf()` function, as Aave tokens are rebasing tokens, meaning the balance will increase over time. By using `balanceOf()` instead of withdrawing everything, gas can be saved by avoiding a subsequent redeposit.

Technical Details

File: AaveVault.sol

```
40:     function _beforeWithdrawal(address _to, uint256 _amount) internal override {
41:         uint128 exited = uint128(aaveV3Pool.withdraw(address(weth),
type(uint256).max, address(this)));
42:
43:         uint256 cachedTotalDeposit = getTotalDeposit();
44:         uint256 interest = exited - cachedTotalDeposit;
45:         uint256 amountToSupply = cachedTotalDeposit - _amount;
46:
47:         if (amountToSupply > 0) {
48:             aaveV3Pool.supply(address(weth), amountToSupply, address(this), 0);
49:         }
50:
51:         weth.withdraw(_amount + interest);
52:
53:         _transfer(address(0), rateReceiver, interest);
54:         _transfer(address(0), _to, _amount);
55:     }
```

Impact

Gas savings.

Recommendation

```
function _beforeWithdrawal(address _to, uint256 _amount) internal override {
        uint256 currentBalance = aEthWETH.balanceOf(address(this));
        uint256 cachedTotalDeposit = getTotalDeposit();
        uint256 interest = currentBalance - cachedTotalDeposit;
        uint256 amountToWithdraw = _amount + interest;

        aaveV3Pool.withdraw(address(weth), amountToWithdraw, address(this))
        weth.withdraw(amountToWithdraw);

        _transfer(address(0), rateReceiver, interest);
        _transfer(address(0), _to, _amount);
    }
```

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/70aa9fc3840bae4f8447b8fef74bfb447694562d.

4. Gas - Reduce storage variables access

Technical Details

Some functions read variables from storage multiple times and could be stored in memory instead:

- In the `apxETHVault` and `AaveVault`, the storage variables `apxETH`, `aaveV3Pool`, and `weth` are read multiple times per function.
- In the function `_lzReceive()` the storage variable `blockMetadata.totalGuessWeigh` and `dripVault` are read multiple times and `lot` can sometimes be updated up to 3 times.
- In `updateDripVault()` the storage variable `dripVault` is read multiple times.
- In `getLatestTail()` the storage variables `minimumBlockAge` and `groupSize` can be read twice.
- In `onValidatorTriggered()` the storage variable `lzEndpointReceiverId` is read twice.

Impact

Gas.

Recommendation

Cache variables and update them only once.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/commit/ebf63da4455772e22ca2208bd7f661b85560881f.

Informational Findings

1. Informational - Unused errors present

Unused error is defined and can be removed.

Technical Details

```
File: src/IGuessOurBlock.sol

11: error RoundNotStarted();

16: error InvalidSender();
```

src/IGuessOurBlock.sol#L11, src/IGuessOurBlock.sol#L16

Impact

Informational

Recommendation

Remove unused code.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/c969050f2989453f6f662d4f537080cf6d5616ea.

2. Informational - Unnecessary cast

Technical Details

The variable is being cast to its own type

```
File: src/GuessOurBlockReceiver.sol

95: lot += uint128(_nativeSent);
```

src/GuessOurBlockReceiver.sol#L95

The cast is not necessary; the variable is used with uint256 afterward.

```
File: src/dripVaults/implementations/AaveVault.sol

41: uint128 exited = uint128(aaveV3Pool.withdraw(address(weth), type(uint256).max,
address(this)));
```

AaveVault.sol#L41-L41

```
File: src/GuessOurBlockReceiver.sol
uint128(Math.mulDiv(_winningLot, _userWeight, _totalGuessWeight));
```

GuessOurBlockReceiver.sol#L258-L258

Impact

Informational

Recommendation

Remove the casting.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/80b1c6c11cb61f4f1147a194f5b168ead203e332.

3. Informational - Unused import

The identifier is imported but never used within the file.

Technical Details

```
File: src/GuessOurBlockSender.sol


5: import { Math } from "@openzeppelin/contracts/utils/math/Math.sol";
```

src/GuessOurBlockSender.sol#L5

Impact

Informational

Recommendation

Remove the unused import.

Developer Response

Resolved hash: fce63494f6505b3a9eed1c26c56fb21195c6aef2

4. Informational - Unused `event` definition

An unused `event` is defined and can be removed.

Technical Details

```
File: src/IGuessOurBlock.sol


28: event RoundPauseTimerUpdated(uint32 pauseTimer);
```

src/IGuessOurBlock.sol#L28

Impact

Informational

Recommendation

Remove the unused event.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/4a19dd3699fc8b601c22dfc5177f03e27fe313d8.

5. Informational - Missing event for state change

Technical Details

```
File: src/GuessOurBlockSender.sol


50: function updateLzGasLimit(uint32 _gasLimit) external onlyOwner {
51:         lzGasLimit = _gasLimit;
52:     }
```

src/GuessOurBlockSender.sol#L50

```
File: GuessOurBlockSender.sol


50:     function updateLzGasLimit(uint32 _gasLimit) external onlyOwner {
51:         lzGasLimit = _gasLimit;
52:     }
```

https://github.com/HeroglyphEVM/GuessOurBlock/blob/30ae3e58017939aa21b79a29c435b68975b960db/src/GuessOurBlockSender.sol#L50-L52

Impact

Informational

Recommendation

Add events to keep track of the changes off-chain.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/a7aea04b9eeff9d682110f9c9150f30fe9a1a0d5.

6. Informational - `GuessOurBlockReceiver` could be deployed on Arbitrum

Technical Details

The contract `GuessOurBlockReceiver` could be deployed on Arbitrum. It is possible to query the current Ethereum block from an Arbitrum contract using `block.number`, as explained in the documentation, it should be accurate with sometimes a small delay but that shouldn't impact the security of the GuessOurBlock protocol.

Deploying on Arbitrum would eliminate the need for a LayerZero integration, simplifying the contract and allowing users to pay lower fees when using the protocol.

Impact

Informational.

Recommendation

Deploy on Arbitrum and remove LayerZero integration.

Developer Response

Acknowledged - Decided to stay on Mainnet.

7. Informational - add a `referralCode` to the `AaveVault`

Technical Details

The `AaveVault` currently uses the code 0 as `referralCode` when supplying `weth`.

AAVE doesn't have an ongoing referral program, but this might change in the future if a proposal is made on the AAVE governance forum. In that case, it will be useful to be able to update the `referralCode` from 0 to an actual code.

Impact

Informational.

Recommendation

Add a `referralCode` storage variable and a setter.

Developer Response

Fixed in https://github.com/HeroglyphEVM/GuessOurBlock/pull/8/commits/6579bd6bf32ffffca4a040891f4516ae8c68ad7c.

Final remarks

The GoB contracts are simple. The team was responsive and quick to reply to questions and comments during the engagement. While the review did not uncover any severe vulnerabilities, auditors caution against the risks posed by the game's current architecture, where users may lose their bet before it becomes active.