shieldify

Guanciale

SECURITY REVIEW

Date: 15 January 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Rust, Go, Vyper, Move and Cairo.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Guanciale

Guanciale AI DAO combines cutting-edge artificial intelligence (AI) with decentralized finance (DeFi) and governance mechanisms to create a robust ecosystem. Leveraging unique approaches through the GambleFi protocols and AI-powered portfolio advisory tools, Guanciale AI DAO aims to redefine decentralized applications by optimizing value accrual, governance transparency, and user engagement. This document outlines the governance model, technical specifications, economic structures, and product-specific roadmaps for establishing a scalable, efficient, and user-centric platform.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors

- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 6 days with a total of 192 hours dedicated by 4 researchers from the Shieldify team.

Overall, the code is well-written. The audit report identified six Critical and High severity issues, five Medium and four Low-severity vulnerabilities, primarily related to the `veGUAN` staking and the wheel game randomness functionality.

The Guanciale team has been very responsive to the Shieldify research team's inquiries, demonstrating a strong commitment to security by taking into account all the recommendations and fixing suggestions from the researchers.

5.1 Protocol Summary

Project Name	**Guanciale**
Repository	guan-staking-contracts
Type of Project	DeFi, Staking
Audit Timeline	6 days
Review Commit Hash	f1f9d252d30edd7e7f04cc536dadae90a9daa509
Fixes Review Commit Hash	f39543b4cfbc2be49cd8b7f9609ae9973d593054

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
src/veGUAN.sol	186
src/games/WheelOfGuantune.sol	275
src/nfts/WheelOfGuantuneNFT.sol	63
src/chainlink/VRFConsumerBaseV2Plus.sol	18
src/chainlink/interfaces/IVRFCoordinatorV2Plus.sol	19
src/chainlink/interfaces/IVRFSubscriptionV2Plus.sol	5
src/chainlink/libraries/VRFV2PlusClient.sol	28
Total	**620**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical/High** issues: **6**
- **Medium** issues: **5**
- **Low** issues: **4**
- **Informational** issues: **6**

ID	Title	Severity	Status
[C-01]	Scaling Of `UD60x18` Variables Sent to `_calculateVotingPower()` Causes Incorrect Calculations	Critical	Fixed
[C-02]	Incorrect Receiver of `Extraspins` and GPoints in `_handlePayout()`	Critical	Fixed
[C-03]	Number of Available Spins Is Never Decreased On Spin or If Spin Is Fulfilled	Critical	Fixed
[H-01]	Setting The Configuration with Arrays of Longer Than 1 Causes Incorrect Config	High	Fixed
[H-02]	Current Implementation of the `fulfillRandomWords()` Might Not Work as Expected	High	Fixed
[H-03]	Users Can Use `Flashloan` to Increase Voting Power of Expired Positions and Execute Proposal for Their Benefits	High	Acknowledged
[M-01]	The `increaseStakeAndLock()` Function Prevents Users from Increasing Stake Amount Only	Medium	Fixed
[M-02]	The Current `veGUAN` Implementation Does Not Give Users Extra Spins Nor the Wheel Contract	Medium	Fixed
[M-03]	Missing Update Function for `wheelOfGuantuneNft` Config Field	Medium	Fixed
[M-04]	Incorrect Import Path in `WheelOfGuantune.sol` Will Cause the Code to Never Compile	Medium	Fixed
[M-05]	Centralization Risks, Especially Changes to Token Address Can Freeze Funds in Contract	Medium	Acknowledged
[L-01]	Delayed Use of `increaseStakeAndLock()` Could Cause Arithmetic Overflow	Low	Fixed
[L-02]	Users Can Have Vote Weight Even When Their Position Expired	Low	Fixed
[L-03]	The `baseURI` Not Being Set in the `initialize()` Function Could Lead To Empty URI Data	Low	Acknowledged
[L-04]	Unused Private State Variable `_veGuanStorage`	Low	Fixed

[I-01]	Unnecessary Check in `getVotingPower()` Function	Informational	Fixed
[I-02]	Current Logic In `_calculateVotingPower()` Will Add `1e18` to the `scalingFactorX18`	Informational	Fixed
[I-03]	Unnecessary Extra Call to Get Storage Variable Value	Informational	Fixed
[I-04]	Current Key Hash for VRF Is Not Correct	Informational	Fixed
[I-05]	User Can Emit Unstake Event Without Unstaking Any Amount	Informational	Fixed
[I-06]	No Need to Do State Updates When The `increaseAndStake()` Function Called with Zero Stake Amount Update	Informational	Acknowledged

7. Findings

[C-01] Scaling Of `UD60x18` Variables Sent to `_calculateVotingPower()` Causes Incorrect Calculations

Severity

Critical Risk

Description

The `_calculateVotingPower()` function uses the `UD60x18` user type from PaulRBerg library. in the example on the site we can see that all inputs are scaled to have `E18` added to them as in UD60x18.

This is not so in the current implementation and the variables sent in from other calls are never scaled correctly, thus the calculation only ever returns the value of the Staked amount no matter what the remaining Stake lock time is.

Location of Affected Code

File: veGUAN.sol#L149

In the function `getVotingPowerOf()`:

```
return _calculateVotingPower(
  ud60x18($.votingPowerCurveAFactor), ud60x18(remainingLockDuration),
     ud60x18(lockedPosition.stake)
);
```

File: veGUAN.sol#L171

In the function `previewNewVotingPower()`:

```
return
  _calculateVotingPower(ud60x18($.votingPowerCurveAFactor), ud60x18(
     newLockDuration), ud60x18(newStakeValue));
}
```

File: veGUAN.sol#L369

```
function _calculateVotingPower(
  UD60x18 votingPowerCurveAFactorX18,
  UD60x18 remainingLockDurationX18,
  UD60x18 positionStakeX18
)
  internal
  pure
  returns (uint256 scalingFactor, uint256 votingPower)
{
  // calculate the lock multiplier as explained in the function's natspec
  UD60x18 scalingFactorX18 = votingPowerCurveAFactorX18.mul(
     remainingLockDurationX18).add(UNIT);

  // return the scaling factor and voting power\
  scalingFactor = scalingFactorX18.intoUint256();
  votingPower = positionStakeX18.mul(scalingFactorX18).intoUint256();
}
```

Impact

The voting power calculations are static to the stake amount and can be manipulated. The voting power also never decays based on remaining lock time.

Proof of Concept

Currently no matter what the remaining time is the voting power is always the ctaked value:

```
[PASS] test_StakeGetVotingPowerSecondStaker() (gas: 441673)
Logs:
  [SecondStakerTest] Locked for Max - 1 week with Stake of 120000000-->
     Voting Power of User1 is: 120000000
  [SecondStakerTest] Locked for Min duration with Stake of 144400000 -->
     Voting Power of User2 is: 144400000
  VM WARP ADDING MIN DURATION + 1
  New values for voting power are:
  [SecondStakerTest] Locked for Max - 1 week with Stake of 120000000-->
     Voting Power of User1 is: 120000000
  [SecondStakerTest] Locked for Min duration with Stake of 144400000 -->
     Voting Power of User2 is: 144400000
```

Code

```
function test_StakeGetVotingPowerSecondStaker() external {
  // it should revert
  vm.startPrank(User1);
  mockLPToken.approve(proxy,120000000);
  VeGuan(proxy).stakeAndMint(120000000, MAX_LOCK_DURATION - 1 weeks);
  vm.stopPrank();
  vm.startPrank(User2);
  mockLPToken.approve(proxy,144400000);
  VeGuan(proxy).stakeAndMint(144400000, MIN_LOCK_DURATION);
  vm.stopPrank();
  (uint256 scalingFactor, uint256 votingPower) = VeGuan(proxy).
     getVotingPowerOf(1);
  console.log("[SecondStakerTest] Locked for Max - 1 week with Stake of
     120000000--> Voting Power of User1 is: %d", votingPower);
  (scalingFactor, votingPower) = VeGuan(proxy).getVotingPowerOf(2);
  console.log("[SecondStakerTest] Locked for Min duration with Stake of
     144400000 --> Voting Power of User2 is: %d", votingPower);
  console.log("VM WARP ADDING MIN DURATION + 1");
  vm.warp(MIN_LOCK_DURATION + 1);
  console.log("New values for voting power are:");
  (scalingFactor, votingPower) = VeGuan(proxy).getVotingPowerOf(1);
  console.log("[SecondStakerTest] Locked for Max - 1 week with Stake of
     120000000--> Voting Power of User1 is: %d", votingPower);
  (scalingFactor, votingPower) = VeGuan(proxy).getVotingPowerOf(2);
  console.log("[SecondStakerTest] Locked for Min duration with Stake of
     144400000 --> Voting Power of User2 is: %d", votingPower);
}
```

Result

Once corrected it gives changed values:

```
[PASS] test_StakeGetVotingPowerSecondStaker() (gas: 443767)
Logs:
  [SecondStakerTest] Locked for Max - 1 week with Stake of 120000000-->
     Voting Power of User1 is:
     847384615244966400012000000000000000000000000
  [SecondStakerTest] Locked for Min duration with Stake of 144400000 -->
     Voting Power of User2 is:
     799753846022046720144400000000000000000000000
  VM WARP ADDING MIN DURATION + 1
  New values for voting power are:
  [SecondStakerTest] Locked for Max - 1 week with Stake of 120000000-->
     Voting Power of User1 is:
     780923076794380800012000000000000000000000000
  [SecondStakerTest] Locked for Min duration with Stake of 144400000 -->
     Voting Power of User2 is: 144400000000000000000000000
```

Recommendation

Scale the values each time the function is called to be `value * 1E18`:

```
function getVotingPowerOf(uint256 tokenId) public view returns (uint256
    scalingFactor, uint256 votingPower) {
  // load veGUAN's storage pointer
  VeGuanStorage storage $ = _getVeGuanStorage();
  // load the locked position's storage pointer
  LockedPositionData storage lockedPosition = _getVeGuanStorage().
     lockedPositions[tokenId];
  uint256 votingCurveCorrected = $.votingPowerCurveAFactor * 1e18;
  // calculate how many seconds are left until the position is unlocked,
     used to determine the voting power
  // note: if the position is unlocked, the following value is set as 0
     which returns a voting power of 0
  uint256 remainingLockDuration =
    block.timestamp > lockedPosition.lockedUntil ? 0 : lockedPosition.
       lockedUntil - block.timestamp;

  remainingLockDuration = remainingLockDuration * 1E18;
  return _calculateVotingPower(
    ud60x18(votingCurveCorrected), ud60x18(remainingLockDuration),
       ud60x18(lockedPosition.stake * 1E18)
  );
}
```

Team Response

Fixed.

[C-02] Incorrect Receiver of `Extraspins` and GPoints in `_handlePayout()`

Severity

Critical Risk

Description

The `_handleRewardPayout()` function is called by `fulfillRandomWords()` which is called by `rawFulfillRandomWords()` in the `VRFConsumerBaseV2Plus` contract. The permissions to call `rawFulfillRandomWords()` is only granted to the `vrfCoordinator`, thus the `msg.sender` can never be the user that needs to receive the extra spins as a reward. The code in the `_handleRewardPayout()` for extra spins and gPoints assigns the spins and points to `msg.sender`.

Location of Affected Code

File: src/games/WheelOfGuantune.sol#L613-L628

`_handleRewardPayout` in `WheelOfGuantune.sol` contract.

```
// code
} else if (rewardType == RewardType.ExtraSpins) {
  // fetch the extra spins amount of the reward id
  uint256 extraSpins = $.config.extraSpinsOfRewardId[$.config.
     extraSpinsOfRewardId.length - 1].get(rewardId);
  // update the user's extra spins balance and store the abi-encoded
     reward value
  $.extraSpinsOfUser[msg.sender] += extraSpins;
  rewardValue = abi.encode(extraSpins);
} else if (rewardType == RewardType.GPoints) {
  // fetch the gPoints amount of the reward id
  uint256 gPoints = $.config.gPointsOfRewardId[$.config.gPointsOfRewardId
     .length - 1].get(rewardId);
  // update the user's gPoints balance
  $.gPointsOfUser[msg.sender] += gPoints;
  // update the total gPoints
  $.totalGPoints = ($.totalGPoints + gPoints).toUint128();
  // store the abi-encoded reward value
  rewardValue = abi.encode(gPoints);
}
// code
```

Impact

The rewarded extra spins are given to the caller which is the `VRFConsumerBaseV2Plus` contract.

Proof of Concept

```
function test_IncorrectAssignSpins() external {
  setConfig();
  vm.prank(veOwner);
  MockWheelOfGuantune(wheelproxy).giveExtraSpins(User1,1);
  uint256 availableSpins = MockWheelOfGuantune(wheelproxy).
     getAvailableSpinsOf(User1);
  console.log("Number of spins avaialble for User1 before first spin : %d
     ",availableSpins);
    vm.prank(User1);
  uint256 vrfRequestId = MockWheelOfGuantune(wheelproxy).spin();
  WheelOfGuantune.SpinRequest memory request = MockWheelOfGuantune(
     wheelproxy).getSpinRequest(vrfRequestId);
  assertEq(request.user, User1);
  assertFalse(request.isFulfilled);
```

```
availableSpins = MockWheelOfGuantune(wheelproxy).getAvailableSpinsOf(
    User1);
console.log("Number of spins avaialble for User1 after first spin : %d",
    availableSpins);
availableSpins = MockWheelOfGuantune(wheelproxy).getAvailableSpinsOf(
    address(mockVRF));
console.log("Number of spins avaialble for mockVRF before fulfil call : %
    d",availableSpins);
mockVRF.doCallBack(wheelproxy,vrfRequestId);
request = MockWheelOfGuantune(wheelproxy).getSpinRequest(vrfRequestId);
assertEq(request.user, User1);
assertTrue(request.isFulfilled);
availableSpins = MockWheelOfGuantune(wheelproxy).getAvailableSpinsOf(
    address(mockVRF));
console.log("Number of spins avaialble for mockVRF after fulfil call : %d
    ",availableSpins);
}
```

Gives output of:

```
[PASS] test_IncorrectAssignSpins() (gas: 626235)
Logs:
  Number of spins avaialble for User1 before first spin : 1
  Number of spins avaialble for User1 after first spin : 1
  Number of spins avaialble for mockVRF before fulfil call : 0
  Number of spins avaialble for mockVRF after fulfil call : 1
```

Recommendation

Consider applying the following changes:

```
} else if (rewardType == RewardType.ExtraSpins) {
  // fetch the extra spins amount of the reward id
  uint256 extraSpins = $.config.extraSpinsOfRewardId[$.config.
      extraSpinsOfRewardId.length - 1].get(rewardId);
  // update the user's extra spins balance and store the abi-encoded
      reward value
- $.extraSpinsOfUser[msg.sender] += extraSpins;
+ $.extraSpinsOfUser[user] += extraSpins;
  rewardValue = abi.encode(extraSpins);
} else if (rewardType == RewardType.GPoints) {
  // fetch the gPoints amount of the reward id
  uint256 gPoints = $.config.gPointsOfRewardId[$.config.gPointsOfRewardId
      .length - 1].get(rewardId);
  // update the user's gPoints balance
- $.gPointsOfUser[msg.sender] += gPoints;
+ $.gPointsOfUser[user] += gPoints;
  // update the total gPoints
}

// code
```

Team Response

Fixed.

[C-03] Number of Available Spins Is Never Decreased On Spin or If Spin Is Fulfilled

Severity

Critical Risk

Description

When a user spins or the user's spin is fulfilled the code has nowhere where it decreases the amount of available spins for a user. However `getAvailableSpinsOf()` is out of scope, thus it may happen elsewhere but in the code, it is never decreased in any way.

Impact

The user has unlimited spins and rewards available to them which can cause the protocol harm.

Proof of Concept

```
function test_DecreasesAllocatedSpin() external {
  setConfig();
  vm.prank(veOwner);
  MockWheelOfGuantune(wheelproxy).giveExtraSpins(User1,1);
  uint256 availableSpins = MockWheelOfGuantune(wheelproxy).
     getAvailableSpinsOf(User1);
  console.log("Number of spins avaialble for User1 before first spin : %d
     ",availableSpins);
    vm.prank(User1);
  uint256 vrfRequestId = MockWheelOfGuantune(wheelproxy).spin();
  WheelOfGuantune.SpinRequest memory request = MockWheelOfGuantune(
     wheelproxy).getSpinRequest(vrfRequestId);
  assertEq(request.user, User1);
  assertFalse(request.isFulfilled);
  availableSpins = MockWheelOfGuantune(wheelproxy).getAvailableSpinsOf(
     User1);
  console.log("Number of spins avaialble for User1 after first spin : %d"
     ,availableSpins);
  mockVRF.doCallBack(wheelproxy,vrfRequestId);
  request = MockWheelOfGuantune(wheelproxy).getSpinRequest(vrfRequestId);
  assertEq(request.user, User1);
  assertTrue(request.isFulfilled);
```

```
//vm.expectRevert();
vm.prank(User1);
vrfRequestId = MockWheelOfGuantune(wheelproxy).spin();
availableSpins = MockWheelOfGuantune(wheelproxy).getAvailableSpinsOf(
   User1);
console.log("Number of spins avaialble for User1 after second spin : %d",
   availableSpins);
console.log("Spin for User1 successful");
}
```

Gives the following output.

```
[PASS] test_DecreasesAllocatedSpin() (gas: 674865)
Logs:
  Number of spins avaialble for User1 before first spin : 1
  Number of spins avaialble for User1 after first spin : 1
  Number of spins avaialble for User1 after second spin : 1
  Spin for User1 successful
```

Recommendation

Implement a decrease in available spin count within the code.

Team Response

Fixed.

[H-01] Setting The Configuration with Arrays of Longer Than 1 Causes Incorrect Config

Severity

High Risk

Description

In the `WheelOfGuantune` contract the expectation is that the configuration will be set using the `configureWheel()` function. The input parameters expect an array of values as below:

```
function configureWheel(
  uint256[] calldata segmentsPerReward,
  RewardType[] calldata rewardTypes,
  bytes[] calldata rewardValues,
  uint256 epochDuration
)
  external
  onlyOwner
```

When the array length of the input is longer than 1 then the loop that runs causes the storage variables to have a new item pushed to the storage variables to accommodate for the new configuration values as below:

```
// iterate over the params to configure the wheel
for (uint256 i; i < expectedParamsLength; i++) {
  // first, get the values of the current iteration

  // note: the reward id is the configuration index + 1
  uint256 rewardId = i + 1;
  uint256 segments = segmentsPerReward[i];
  RewardType rewardType = rewardTypes[i];
  bytes memory rewardValue = rewardValues[i];

  // create new UintToUintMap instances for the new rewards config,
     resetting the previously configured values
  $.config.rewardTypeOfRewardId.push();
  $.config.rewardIdOfSegment.push();
  $.config.tokenIdOfRewardId.push();
  $.config.extraSpinsOfRewardId.push();
  $.config.gPointsOfRewardId.push();

  // code
}
```

The problem with this is that instead of each iteration being added to the new config record, each iteration creates it's own new config record, ie:

```
Imagine 2 items in array
Iteration 1:
  $.config.rewardTypeOfRewardId[0].set  one entry with value of=====> a
     rewardId of the value 1.
Iteration 2:
  $.config.rewardTypeOfRewardId[1].set one entry with value of=====> a
     rewardId of the value 2.

Instead of:
Iteration 1:
  $.config.rewardTypeOfRewardId[0].set  FIRST ENTRY=====> a rewardId of
     the value 1.
Iteration 2:
  $.config.rewardTypeOfRewardId[0].set  SECOND ENTRY=====> a rewardId of
     the value 2.
```

Location of Affected Code

File: src/games/WheelOfGuantune.sol#L324

```
function configureWheel(
  uint256[] calldata segmentsPerReward,
  RewardType[] calldata rewardTypes,
  bytes[] calldata rewardValues,
  uint256 epochDuration
)
  external
  onlyOwner
{
  // code

  // iterate over the params to configure the wheel
  for (uint256 i; i < expectedParamsLength; i++) {
    // first, get the values of the current iteration

    // note: the reward id is the configuration index + 1
    uint256 rewardId = i + 1;
    uint256 segments = segmentsPerReward[i];
    RewardType rewardType = rewardTypes[i];
    bytes memory rewardValue = rewardValues[i];
```

```
    // create new UintToUintMap instances for the new rewards config,
       resetting the previously configured values
    $.config.rewardTypeOfRewardId.push();
    $.config.rewardIdOfSegment.push();
    $.config.tokenIdOfRewardId.push();
    $.config.extraSpinsOfRewardId.push();
    $.config.gPointsOfRewardId.push();

    // code
  }

  // code
}
```

Impact

The `getRewards()` function is rendered non-functional, and the reward types are set incorrectly.

Proof of Concept

```
function test_OverwriteConfig() external {
  setBiggerConfig();
  WheelOfGuantune.Reward[] memory rewards = MockWheelOfGuantune(
     wheelproxy).getWheelRewards();
  for(uint256 i;i<rewards.length;i++){
    console.log("in loop");
    WheelOfGuantune.Reward memory tmpReward = rewards[i];
    console.log("[%d] rewardType is: %d",i,uint256(tmpReward.rewardType))
       ;
    console.log("[%d] value is: ",i);
    console.logBytes(tmpReward.value);
  }
  setBiggerConfig();
  WheelOfGuantune.Reward[] memory rewardsnew = MockWheelOfGuantune(
     wheelproxy).getWheelRewards();
  for(uint256 i;i<rewardsnew.length;i++){
    WheelOfGuantune.Reward memory tmpReward = rewardsnew[i];
    console.log("[%d] rewardType is: %d",i,uint256(tmpReward.rewardType))
       ;
    console.log("[%d] value is: ",i);
    console.logBytes(tmpReward.value);
  }
  setBiggerConfig();
  rewardsnew = MockWheelOfGuantune(wheelproxy).getWheelRewards();
  for(uint256 i;i<rewardsnew.length;i++){
    WheelOfGuantune.Reward memory tmpReward = rewardsnew[i];
    console.log("[%d] rewardType is: %d",i,uint256(tmpReward.rewardType))
       ;
    console.log("[%d] value is: ",i);
    console.logBytes(tmpReward.value);
  }
  console.log("Reward array length is: %d",rewardsnew.length);
}
```

```
function setBiggerConfig() public {
  uint256[] memory probabilityPerReward = new uint256[](2);
  uint256[] memory segmentsPerReward = new uint256[](2);
  WheelOfGuantune.RewardType[] memory rewardTypes = new WheelOfGuantune.
     RewardType[](2);
  bytes[] memory rewardValues = new bytes[](2);
  uint256 epochDuration = uint256(2 days);
  probabilityPerReward[0] = 15;
  segmentsPerReward[0] = 3;
  rewardTypes[0] = WheelOfGuantune.RewardType.GPoints;
  rewardValues[0] = abi.encode(8);

  probabilityPerReward[1] = 12;
  segmentsPerReward[1] = 2;
  rewardTypes[1] = WheelOfGuantune.RewardType.Nft;
  rewardValues[1] = abi.encode(5);

  vm.prank(veOwner);
  MockWheelOfGuantune(wheelproxy).configureWheel(segmentsPerReward,
    rewardTypes,
    rewardValues,
    epochDuration);
}
```

Recommendation

Move the code that pushes new values outside the loop to only run once per call to `configureWheel()`:

```
function configureWheel(
  uint256[] calldata probabilityPerReward,
  uint256[] calldata segmentsPerReward,
  RewardType[] calldata rewardTypes,
  bytes[] calldata rewardValues,
  uint256 epochDuration
)
  external
  onlyOwner
{
  // code
```

```
  // prepare the new total segments value
  uint256 totalSegments;

+ ====> MOVE CODE HERE
+ // create new UintToUintMap instances for the new rewards config,
   resetting the previously configured values
+ $.config.rewardTypeOfRewardId.push();
+ $.config.rewardIdOfSegment.push();
+ $.config.tokenIdOfRewardId.push();
+ $.config.extraSpinsOfRewardId.push();
+ $.config.gPointsOfRewardId.push();

  // iterate over the params to configure the wheel
  for (uint256 i; i < expectedParamsLength; i++) {
    // first, get the values of the current iteration
    @@ -309,13 +317,6 @@ contract WheelOfGuantune is
    RewardType rewardType = rewardTypes[i];
    bytes memory rewardValue = rewardValues[i];

-   // create new UintToUintMap instances for the new rewards config,
   resetting the previously configured values
-   $.config.rewardTypeOfRewardId.push();
-   $.config.rewardIdOfSegment.push();
-   $.config.tokenIdOfRewardId.push();
-   $.config.extraSpinsOfRewardId.push();
-   $.config.gPointsOfRewardId.push();

    // now, populate the new maps

    // cast to uint256 to store the enum value

    // code
  }
}
```

Team Response

Fixed.

[H-02] Current Implementation of the `fulfillRandomWords()` Might Not Work as Expected

Severity

High Risk

Description

The `fulfillRandomWords()` function is meant to return the reward for users who called spin to get a VRF id, this function will determine the reward of the NFT depending on the `totalSegments` which

is equal to all segments added so far, the reward specifying mechanism is as below:

```
uint256 randomWord = randomWords[0];

// fetch the EIP-7201 storage slot
WheelOfGuantuneStorage storage $ = _getWheelOfGuantuneStorage();

// calculate the winning segment based on the random word
// note: we add 1 as the first segment is 1 and the last segment is the
   total segments value of the wheel
uint256 winningSegment = randomWord % $.totalSegments + 1;

// fetch the reward id of the winning segment
uint256 rewardId = $.config.rewardIdOfSegment[$.config.rewardIdOfSegment.
   length - 1].get(winningSegment);

// fetch the reward type of the winning segment
RewardType rewardType =
  RewardType($.config.rewardTypeOfRewardId[$.config.rewardTypeOfRewardId.
     length - 1].get(rewardId));

// fetch the user's spin request and cache its address
SpinRequest storage spinRequest = $.spinRequests[requestId];
address user = spinRequest.user;

// update the user's state
$.isSpinning[user] = false;
spinRequest.isFulfilled = true;
$.epochSpinsSpentPerUser[_getCurrentEpoch()][user]++;

// handle the reward based on the reward type
bytes memory rewardValue = _handleRewardPayout(user, rewardId, rewardType
   );
```

As shown above, we mod the word returned by VRF to the number of the `totalSegments`, then we use the returned number to get the reward id and then the reward type, however, there is an issue in this implementation, we can see that the `rewardId` is returned using the latest mapping that added when the `configureWheel()` function gets called, the last mapping might have less element compared to the `winningSegment` that returned. for example:

- $.config.rewardTypeOfRewardId.length - 1 = 2 elements
- the \codex{randomWord % $.totalSegments + 1;} equals to \codex{15 % 15 +1 == 3.25 and rounding it down to 3}
- when we give an index of 3 to a mapping that have 2 elements only, this will return a null mapping element which leads to returning zero rewards for users.

We can see that each time the `config()` function is called, a new mapping gets added and the `totalSegment` equal to all mapping segments if we say 3 mappings get added and the first one = 11 segments and the second and third are equal to 2 segments then the `totalSegment` is 15 and the

last mapping that gets used in `fulfillRandomWords()` contains 2 elements only, if we assume the word = 15 and mod it to 15 and then add 1 to it will return 3 which is a null element in the last mapping.

Location of Affected Code

File: src/games/WheelOfGuantune.sol#L531

```
function fulfillRandomWords(uint256 requestId, uint256[] calldata
    randomWords) internal override {
  // get the random word, which is the first element of the random words
     array as we only requested one word
  uint256 randomWord = randomWords[0];

  // fetch the EIP-7201 storage slot
  WheelOfGuantuneStorage storage $ = _getWheelOfGuantuneStorage();

  // calculate the winning segment based on the random word
  // note: we add 1 as the first segment is 1 and the last segment is the
      total segments value of the wheel
  uint256 winningSegment = randomWord % $.totalSegments + 1;

  // fetch the reward id of the winning segment
  uint256 rewardId = $.config.rewardIdOfSegment[$.config.
     rewardIdOfSegment.length - 1].get(winningSegment);

  // fetch the reward type of the winning segment
  RewardType rewardType =
    RewardType($.config.rewardTypeOfRewardId[$.config.
       rewardTypeOfRewardId.length - 1].get(rewardId));

  // fetch the user's spin request and cache its address
  SpinRequest storage spinRequest = $.spinRequests[requestId];
  address user = spinRequest.user;
```

```
    // update the user's state
    $.isSpinning[user] = false;
    spinRequest.isFulfilled = true;
    $.epochSpinsSpentPerUser[_getCurrentEpoch()][user]++;

    // handle the reward based on the reward type
    bytes memory rewardValue = _handleRewardPayout(user, rewardId,
        rewardType);

    emit LogSpinRequestFulfilled(user, rewardId, requestId, rewardType,
        rewardValue, winningSegment);
}

/// @dev EIP-7102 storage slot getter.
function _getWheelOfGuantuneStorage() internal pure returns (
    WheelOfGuantuneStorage storage $) {
    bytes32 slot = WheelOfGuantuneStorageLocation;

    assembly {
      $.slot := slot
    }
}
```

Impact

Users might get no rewards for the requested spins.

Recommendation

If the protocol team accepted this bug, then the code should be highly modified to implement the correct logic of the wheel contract, currently, the fix is unknown to me since the fulfillRandomWords should be re-implemented in case of validating this report.

Team Response

Fixed.

[H-03] Users Can Use `Flashloan` to Increase Voting Power of Expired Positions and Execute Proposal for Their Benefits

Severity

High Risk

Description

If we assume the Medium-01 from the report is fixed in the `increaseAndStake()` function which allows users to add the amount to their stake without updating the lock duration then the below scenario might be executable:

- Assume Alice's `lockUntil` reached the current `block.timestamp`.

- Alice got a huge flashloan of GUAN token (can get another token as flashloan and then swap it to GUAN) and called the `increaseAndStake()` function with the flashloan amount.
- If we assume the Medium-01 issue is fixed then the transaction will be executed without reverting since Alice increased the stake amount only.
- The `veGUAN` core logic allows the stakes to have voting power depending on their stake amount even if the lock duration expired, this is clearly shown in the function below:

```
function _calculateVotingPower(
  UD60x18 votingPowerCurveAFactorX18,
  UD60x18 remainingLockDurationX18,
  UD60x18 positionStakeX18
)
  internal
  pure
  returns (uint256 scalingFactor, uint256 votingPower)
{
  // calculate the lock multiplier as explained in the function's natspec
  UD60x18 scalingFactorX18 = votingPowerCurveAFactorX18.mul(
     remainingLockDurationX18).add(UNIT); // @audit 1e18 get added even
     if the calc = 0

  // return the scaling factor and voting power
  scalingFactor = scalingFactorX18.intoUint256();
  votingPower = positionStakeX18.mul(scalingFactorX18).intoUint256();
}
```

- This way Alice can have huge voting power due to her flashloan amount and she can execute a proposal and vote for it in one transaction and then unstake her GUAN token(the tx won't revert since `block.timestamp == lockUntil`:

```
function unstake(uint256 tokenId, uint256 amount) external onlyTokenOwner
    (tokenId) {
  // load veGUAN storage slot
  VeGuanStorage storage $ = _getVeGuanStorage();

  // load the lock data storage pointer
  LockedPositionData storage lockedPosition = $.lockedPositions[tokenId];

  // revert if the position is still locked
  if (block.timestamp < lockedPosition.lockedUntil) {
      revert PositionIsLocked();
  }

  // deduct the unstake amount from the locked position's state, if there
      isn't enough stake in the position the
  // call will revert with an underflow
  lockedPosition.stake -= amount;

  // transfer the lp tokens to the `msg.sender`
  IERC20($.lpToken).safeTransfer(msg.sender, amount);

  // cache the veGUAN's voting power
  (, uint256 votingPower) = getVotingPowerOf(tokenId);

  // emit an event
  emit LogUnstake(msg.sender, tokenId, lockedPosition.stake, votingPower)
      ;
}
```

This issue could potentially occur based on the current small codebase. However, the GUAN documentation states that proposals are reviewed by the council, which may prevent this issue from being executed.

Location of Affected Code

File: src/veGUAN.sol#L283

```
/// @notice Increases the stake value and/or lock duration of a veGUAN
    position.
/// @param tokenId The NFT identifier.
/// @param stakeIncrease The amount of LP tokens to be staked into the
    position.
/// @param lockIncrease The amount of time to add to the position's lock
    duration.
function increaseStakeAndLock(
  uint256 tokenId,
  uint256 stakeIncrease,
  uint256 lockIncrease
)
  external
  onlyTokenOwner(tokenId)
{
  // load veGUAN storage slot
  VeGuanStorage storage $ = _getVeGuanStorage();

  // load the lock data storage pointer
  LockedPositionData storage lockedPosition = $.lockedPositions[tokenId];

  // cache the new unlock timestamp value
  uint256 newUnlockTimestamp = lockedPosition.lockedUntil + lockIncrease;

  // compute the new lock duration based on the provided lockIncrease
  uint256 newLockDuration = newUnlockTimestamp - block.timestamp;

  // validate that the new lock duration is under the min and max
     requirements
  if (newLockDuration > $.maxLockDuration || newLockDuration < $.
     minLockDuration) {
      revert InvalidLockDuration();
  }

  uint256 newStakeValue = lockedPosition.stake + stakeIncrease;

  // updates the locked position data
  lockedPosition.stake = newStakeValue;
  lockedPosition.lockedUntil = newUnlockTimestamp;
  // finally, transfer the lp tokens from the `msg.sender`
  IERC20($.lpToken).safeTransferFrom(msg.sender, address(this),
     stakeIncrease);

  // cache the veGUAN's voting power
  (, uint256 votingPower) = getVotingPowerOf(tokenId);

  // emit an event
  emit LogIncreaseStakeAndLock(msg.sender, tokenId, newStakeValue,
     newUnlockTimestamp, votingPower);
}
```

File: src/veGUAN.sol#L327

```
/// @notice Unstakes a given amount of LP tokens from an unlocked veGUAN
    position.
/// @param tokenId The NFT identifier.
/// @param amount The amount of LP tokens to unstake from the veGUAN
    position.
function unstake(uint256 tokenId, uint256 amount) external onlyTokenOwner
    (tokenId) {
  // load veGUAN storage slot
  VeGuanStorage storage $ = _getVeGuanStorage();

  // load the lock data storage pointer
  LockedPositionData storage lockedPosition = $.lockedPositions[tokenId];

  // revert if the position is still locked
  if (block.timestamp < lockedPosition.lockedUntil) {
      revert PositionIsLocked();
  } // @audit

  // deduct the unstake amount from the locked position's state, if there
      isn't enough stake in the position the
  // call will revert with an underflow
  lockedPosition.stake -= amount;

  // transfer the lp tokens to the `msg.sender`
  IERC20($.lpToken).safeTransfer(msg.sender, amount);

  // cache the veGUAN's voting power
  (, uint256 votingPower) = getVotingPowerOf(tokenId);

  // emit an event
  emit LogUnstake(msg.sender, tokenId, lockedPosition.stake, votingPower)
      ;
}
```

Impact

A malicious user can execute a flashloan attack to gain huge vote power to execute a proposal.

Recommendation

If the check changed from the `unstake()` function then the attack can be prevented:

```
if (block.timestamp <= lockedPosition.lockedUntil) {
```

Another check can be added in `increaseAndStake()` which prevents increasing stake amount for expired positions.

Team Response

Acknowledged.

[M-01] The `increaseStakeAndLock()` Function Prevents Users from Increasing Stake Amount Only

Severity

Medium Risk

Description

The function `increaseStakeAndLock()` is meant to allow users to increase stake amount AND/OR increase their lock durations, however, this is not how the current logic works in the `increaseStakeAndLock()` function, the logic implemented in this function prevents users who want to increase their stake amount only to invoke it, this is because the function has the check below:

```
// validate that the new lock duration is under the min and max
   requirements
if (newLockDuration > $.maxLockDuration || newLockDuration < $.
   minLockDuration) {
  revert InvalidLockDuration();
} //@audit If we didn't add a lock then this function reverts
```

Let's assume the scenario below:

- Bob called the `stakeAndMint()` function by setting the lock duration to 10 days with 100 GUAN tokens.
- Now Bob's state is like this: `lockedUntil`: 10 days and `stake`: 100 GUAN
- The minimum lock duration is set to 7 days by the contract owner.
- Now Bob wants to increase his stake amount by adding 50 GUAN tokens to his stake balance without modifying the lock duration.

The Bob transaction will revert because of the lines below:

```
// cache the new unlock timestamp value
uint256 newUnlockTimestamp = lockedPosition.lockedUntil + lockIncrease;
   //@audit since lockIncrease == 0 this will return the 10 days in block
   .timestamp that bob set when staked tokens first time

// compute the new lock duration based on the provided lockIncrease
uint256 newLockDuration = newUnlockTimestamp - block.timestamp; //@audit
   since this function called after 5 days after the first stake the
   newLockDuration will be equal to 5 days(in block.timestamp)

// validate that the new lock duration is under the min and max
   requirements
if (newLockDuration > $.maxLockDuration || newLockDuration < $.
   minLockDuration) {
  revert InvalidLockDuration();
} //@audit since 5 days(newLockDuration) is smaller than 7 days (
   minLockDuration) the transaction will revert.
```

As explained in the code above, since 5 days have passed since Bob staked for the first time, the current `newLockDuration` is equal to 5 days which is smaller than the min duration, this will revert the function call and prevent Bob from adding the amount and will force bob to increase its lock duration too which is not the intended behaviour of the current function:

```
/// @notice Increases the stake value and / or lock duration of a veGUAN
    position. @audit
/// @param tokenId The NFT identifier.
/// @param stakeIncrease The amount of LP tokens to be staked into the
    position.
/// @param lockIncrease The amount of time to add to the position's lock
    duration.
function increaseStakeAndLock(
  uint256 tokenId,
  uint256 stakeIncrease,
  uint256 lockIncrease
)
  external
  onlyTokenOwner(tokenId)
{
```

Location of Affected Code

File: src/veGUAN.sol#L283

```
/// @notice Increases the stake value and / or lock duration of a veGUAN
    position. @audit
/// @param tokenId The NFT identifier.
/// @param stakeIncrease The amount of LP tokens to be staked into the
    position.
/// @param lockIncrease The amount of time to add to the position's lock
    duration.
function increaseStakeAndLock(
  uint256 tokenId,
  uint256 stakeIncrease,
  uint256 lockIncrease
)
  external
  onlyTokenOwner(tokenId)
{
  // load veGUAN storage slot
  VeGuanStorage storage $ = _getVeGuanStorage();

  // load the lock data storage pointer
  LockedPositionData storage lockedPosition = $.lockedPositions[tokenId];

  // cache the new unlock timestamp value
  uint256 newUnlockTimestamp = lockedPosition.lockedUntil + lockIncrease;

  // compute the new lock duration based on the provided lockIncrease
  uint256 newLockDuration = newUnlockTimestamp - block.timestamp;

  // validate that the new lock duration is under the min and max
     requirements
  if (newLockDuration > $.maxLockDuration || newLockDuration < $.
     minLockDuration) {
      revert InvalidLockDuration();
  }

  uint256 newStakeValue = lockedPosition.stake + stakeIncrease;

  // updates the locked position data
  lockedPosition.stake = newStakeValue;
  lockedPosition.lockedUntil = newUnlockTimestamp;

  // finally, transfer the lp tokens from the `msg.sender`
  IERC20($.lpToken).safeTransferFrom(msg.sender, address(this),
     stakeIncrease);

  // cache the veGUAN's voting power
  (, uint256 votingPower) = getVotingPowerOf(tokenId);

  // emit an event
  emit LogIncreaseStakeAndLock(msg.sender, tokenId, newStakeValue,
     newUnlockTimestamp, votingPower);
}
```

Impact

The function `increaseStakeAndLock()` prevents users who want to increase their stake amount from invoking the function because a logical error exists in the `increaseStakeAndLock()` function.

Recommendation

It is recommended to bypass the lock duration check when the lock duration is not being updated. This allows users to add to their stake without requiring them to update the lock duration.

Additionally, expiry should be considered, as the current implementation does not include an expiry check when users stake or increase their stake amount/duration.

Team Response

Fixed.

[M-02] The Current `veGUAN` Implementation Does Not Give Users Extra Spins Nor the Wheel Contract

Severity

Medium Risk

Description

The current implementation of the wheel contract allows users with valid `extraSpins` to execute calls to the spin function, the `extraSpin` should be increased when users have a stake in `veGUAN` contract, but currently, none of the `veGUAN` or wheel contract does not have a logic to increase specific user spins to allow them to call the spin function. this way spin is not executable until the user has a valid spin.

Location of Affected Code

File: src/games/WheelOfGuantune.sol#L470

```
/// @notice Spin the wheel to potentially earn a reward.
/// @dev Enforced Invariants:
///      The caller MUST NOT be spinning.
///      The caller MUST have spins available.
function spin() external whenNotPaused nonReentrant returns (uint256
    vrfRequestId) {
  // fetch the EIP-7201 storage slot
  WheelOfGuantuneStorage storage $ = _getWheelOfGuantuneStorage();

  // revert if the wheel hasn't been configured yet
  if (!$.config.isConfigured) revert WheelIsNotConfigured();

  // if user is already spinning, revert
  if ($.isSpinning[msg.sender]) revert UserAlreadySpinning();

  uint256 currentEpoch = _getCurrentEpoch();

  // if user has no spins available, revert
  if (getAvailableSpinsOf(msg.sender) == 0) revert NoAvailableSpins(); //
      @audit always revert

  // cache the chainlink VRF coordinator contract
  IVRFCoordinatorV2Plus vrfCoordinator = vrfCoordinator();

  // submit the random words request to the VRF coordinator
  vrfRequestId = vrfCoordinator.requestRandomWords(
    VRFV2PlusClient.RandomWordsRequest({
      keyHash: VRF_KEY_HASH,
      subId: $.config.vrfSubscriptionId,
      requestConfirmations: VRF_REQUEST_CONFIRMATIONS,
      callbackGasLimit: VRF_CALLBACK_GAS_LIMIT,
      numWords: VRF_NUM_WORDS,
      extraArgs: VRFV2PlusClient._argsToBytes(VRFV2PlusClient.ExtraArgsV1
         ({ nativePayment: VRF_USE_NATIVE }))
    })
  );

  // update the user's state
  $.isSpinning[msg.sender] = true;
  $.spinRequests[vrfRequestId] = SpinRequest({ user: msg.sender,
     isFulfilled: false });

  emit LogSpin(msg.sender, currentEpoch, vrfRequestId);
}
```

Impact

Users can not spin since there is no function that gives them spins.

Recommendation

Add a logic or function that gives users spins if they deserve it by staking their GUAN/LP.

Team Response

Fixed.

[M-03] Missing Update Function for `wheelOfGuantuneNft` Config Field

Severity

Medium Risk

Description

All other config fields have functions or ways to update them except the `wheelOfGuantuneNft` config field.

Impact

All other config items can be updated except the `wheelOfGuantuneNft` config field.

Recommendation

```
+   function updateWheelOfGuantuneNft(address _wheelOfGuantuneNft)
    external onlyOwner {
+     if (_wheelOfGuantuneNft == address(0)) revert ZeroAddress();
+     _getWheelOfGuantuneStorage().config.wheelOfGuantuneNft =
    _wheelOfGuantuneNft;
+   }
```

Team Response

Fixed.

[M-04] Incorrect Import Path in `WheelOfGuantune.sol` Will Cause the Code to Never Compile

Severity

Medium Risk

Description

The import path of the code does not follow the same naming convention as the actual file:

```
import { VeGuan } from "src/VeGuan.sol";
```

However, the actual file name is: `src/veGUAN.sol`

Location of Affected Code

File: src/games/WheelOfGuantune.sol#L17

Impact

The code will not compile.

Recommendation

Correct the import path.

Team Response

Fixed.

[M-05] Centralization Risks, Especially Changes to Token Address Can Freeze Funds in Contract

Severity

Medium Risk

Description

Contracts are controlled by owners with privileged rights to perform administrative tasks, which requires trusting them not to make malicious updates. One potential risk is freezing funds in the contract by altering the LP token address.

Consider the following scenario:

- Users stake their LP/GUAN tokens, locking them in the `veGuan` contract.
- Over time, as the contract accumulates a significant amount of LP/GUAN tokens, the owner changes the LP token address to a worthless token.
- The owner then transfers this new, worthless LP token to the veGUAN contract.
- As a result, when users attempt to unstake, they receive the worthless LP token, losing their valuable GUAN tokens, which remain frozen.

Location of Affected Code

File: src/veGUAN.sol

```
function setBaseURI(string calldata baseURI) external onlyOwner {

function setLpToken(address lpToken) external onlyOwner {

function setVotingPowerCurveAFactor(uint256 votingPowerCurveAFactor)
    external onlyOwner {

function setLockTimeLimits(uint128 minLockDuration, uint128
    maxLockDuration) external onlyOwner {
```

Impact

Should the LP token address be changed any user unstaking would potentially receive a totally different token, with a different value to the original.

Other changes could impact how the protocol parameters are set, also changing the voting power of users.

Recommendation

Consider adding a Timelock or/and using a multisig.

Team Response

Acknowledged.

[L-01] Delayed Use of `increaseStakeAndLock()` Could Cause Arithmetic Overflow

Severity

Low Risk

Description

If a user waits too long before calling `increaseStakeAndLock()`, it may cause an arithmetic overflow. The function adds the new lock value to the old lock value and then subtracts the current timestamp from the result.

If a user unstakes after waiting too long or leaves their stake in the contract for an extended period, an arithmetic overflow may occur, allowing them to only unstake.

Location of Affected Code

File: src/veGUAN.sol#L301

```
uint256 newUnlockTimestamp = lockedPosition.lockedUntil + lockIncrease;

// compute the new lock duration based on the provided lockIncrease
uint256 newLockDuration = newUnlockTimestamp - block.timestamp;
```

Impact

The user will need to either complete unstake if they still have locked funds and then call `stakeAndMint()` for a new tokenId, or if there are no locked funds they would need to call `stakeAndMint()` also for a new `tokenId`.

Proof of Concept

Code

```
function test_StakeIncreaseStakeTooFarInFuture() external {
  vm.startPrank(User1);
  mockLPToken.approve(proxy,1100);
  VeGuan(proxy).stakeAndMint(1000, MIN_LOCK_DURATION+1);
  (uint256 scalingFactor, uint256 votingPower) = VeGuan(proxy).
     getVotingPowerOf(1);
  console.log("[StakeIncreaseStakeSuccess] Voting Power is: %d",
     votingPower);
  vm.warp(MAX_LOCK_DURATION);
  VeGuan(proxy).increaseStakeAndLock(1,0,MIN_LOCK_DURATION);
  vm.stopPrank();
}
```

Result

```
[FAIL: panic: arithmetic underflow or overflow (0x11)]
   test_StakeIncreaseStakeTooFarInFuture() (gas: 264314)
Logs:
  [StakeIncreaseStakeSuccess] Voting Power is:
     553846382692608900100000000000000000000

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 5.52ms
    (315.89s CPU time)

Ran 1 test suite in 11.11ms (5.52ms CPU time): 0 tests passed, 1 failed,
   0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/stakeAndMint/stakeAndMint.t.sol:
   StakeAndMint_Test
[FAIL: panic: arithmetic underflow or overflow (0x11)]
   test_StakeIncreaseStakeTooFarInFuture() (gas: 264314)
```

Recommendation

Change the code to use the current timepstamp if the new duration value is added to the current locked duration as below:

```
// cache the new unlock timestamp value
uint256 newUnlockTimestamp = lockedPosition.lockedUntil + lockIncrease;

+ if(newUnlockTimestamp < block.timestamp){
+    newUnlockTimestamp = block.timestamp + lockIncrease;
+ }

// compute the new lock duration based on the provided lockIncrease
uint256 newLockDuration = newUnlockTimestamp - block.timestamp;
```

Team Response

Fixed.

[L-02] Users Can Have Vote Weight Even When Their Position Expired

Severity

Low Risk

Description

The voting weight mechanism allows users who have expired lock duration to have vote weight even when their position expired, this is because of the way the function `getVotingPowerOf()` calculates the voting power:

File: src/veGUAN.sol#L149

```
function getVotingPowerOf(uint256 tokenId) public view returns (uint256
    scalingFactor, uint256 votingPower) {
  // load veGUAN's storage pointer
  VeGuanStorage storage $ = _getVeGuanStorage();
  // load the locked position's storage pointer
  LockedPositionData storage lockedPosition = _getVeGuanStorage().
     lockedPositions[tokenId];

  // calculate how many seconds are left until the position is unlocked,
     used to determine the voting power
  // note: if the position is unlocked, the following value is set as 0
     which returns a voting power of 0
  uint256 remainingLockDuration =
    block.timestamp > lockedPosition.lockedUntil ? 0 : lockedPosition.
       lockedUntil - block.timestamp; //if expired then return 0

  return _calculateVotingPower(
    ud60x18($.votingPowerCurveAFactor), ud60x18(remainingLockDuration),
       ud60x18(lockedPosition.stake)
  );
}
```

The function above will return zero for `remainingLockDuration` when the lock is expired `block.timestamp > lockedPosition.lockedUntil` however when the `_calculateVotingPower()` get called the logic below executed:

File: src/veGUAN.sol#L369

```
function _calculateVotingPower(
  UD60x18 votingPowerCurveAFactorX18,
  UD60x18 remainingLockDurationX18,
  UD60x18 positionStakeX18
)
  internal
  pure
  returns (uint256 scalingFactor, uint256 votingPower)
{
  // calculate the lock multiplier as explained in the function's natspec
  UD60x18 scalingFactorX18 = votingPowerCurveAFactorX18.mul(
     remainingLockDurationX18).add(UNIT); // @audit 1e18 get added even
     if the calc = 0

  // return the scaling factor and voting power
  scalingFactor = scalingFactorX18.intoUint256();
  votingPower = positionStakeX18.mul(scalingFactorX18).intoUint256();
}
```

The `_calculateVotingPower()` function adds 1e18 to the `scalingFactorX18`, which means the `votingPower` will return a value of the stake multiplied by 1e18, even if the position has expired. This behaviour aligns with the expected logic of the codebase, as described in the following NatSpec:

$$f(x) = a * x + 1 | x \in [0, 52]$$

However, we believe that expired positions should not have voting power until they update their lock duration.

Location of Affected Code

File: src/veGUAN.sol#L149

File: src/veGUAN.sol#L369

Impact

Users with expired positions still have some voting weight power.

Recommendation

We recommend restricting users with expired positions from having voting power unless this is an intentional and acknowledged behaviour of the protocol.

Team Response

Fixed.

[L-03] The `baseURI` Not Being Set in the `initialize()` Function Could Lead To Empty URI Data

Severity

Low Risk

Description

The `initialize()` function does not set the `baseURI` when it is invoked, if the user mint NFT after the `initialize()` function gets invoked then the user gets an NFT with an empty URI, this might cause trouble when the user plans to sell the NFT on the open market.

Location of Affected Code

File: veGUAN.sol#L95

```
function initialize(
  address owner,
  address lpToken,
  uint128 votingPowerCurveAFactor,
  uint128 minLockDuration,
  uint128 maxLockDuration
)
  external
  initializer
{
  ///@dev, note: you can use if clause rather than require check(optional
     )
  require(owner != address(0), "set owner correctly");
  require(lpToken != address(0), "LP token is not valid");
  require(votingPowerCurveAFactor != 0, "votingPowerCurveAFactor is not
     valid");
  require(minLockDuration != 0 , "minLockDuration is not valid");
  require(maxLockDuration != 0 , "maxLockDuration is not valid");

  __ERC721_init("vote-escrowed GUAN", "veGUAN");
  __Ownable_init(owner);

  VeGuanStorage storage $ = _getVeGuanStorage();

  $.lpToken = lpToken;
  $.votingPowerCurveAFactor = votingPowerCurveAFactor;
  $.minLockDuration = minLockDuration;
  $.maxLockDuration = maxLockDuration;
}
```

Impact

The URI for NFTs is not set during the initialization process, if the user mints NFT after the `initialize()` function is invoked, the user will get an empty URI NFT.

Recommendation

Consider applying the following changes:

```
function initialize(
  address owner,
  address lpToken,
  uint128 votingPowerCurveAFactor,
  uint128 minLockDuration,
+ string BaseURISet,
  uint128 maxLockDuration
)
  external
  initializer
{
  ///@dev, note: you can use the if clause rather than require check(
     optional)
  require(owner != address(0), "set owner correctly");
  require(lpToken != address(0), "LP token is not valid");
  require(votingPowerCurveAFactor != 0, "votingPowerCurveAFactor is not
     valid");
  require(minLockDuration != 0 , "minLockDuration is not valid");
  require(maxLockDuration != 0 , "maxLockDuration is not valid");

  __ERC721_init("vote-escrowed GUAN", "veGUAN");
  __Ownable_init(owner);

  VeGuanStorage storage $ = _getVeGuanStorage();

  $.lpToken = lpToken;
  $.votingPowerCurveAFactor = votingPowerCurveAFactor;
  $.minLockDuration = minLockDuration;
  $.maxLockDuration = maxLockDuration;
+ $.baseURI = BaseURISet;
}
```

Team Response

Acknowledged.

[L-04] Unused Private State Variable `_veGuanStorage`

Severity

Low Risk

Description

Unused private state variable `_veGuanStorage`.

Location of Affected Code

File: src/veGUAN.sol#L46

```
VeGuanStorage private _veGuanStorage;
```

Impact

Increased gas cost.

Recommendation

Remove unused variable declaration.

Team Response

Fixed.

[I-01] Unnecessary Check in `getVotingPower()` Function

Severity

Informational

Description

After adding the `lockedUntil <= block.timestamp` check in the function, the following line is no longer needed because, if `block.timestamp` exceeds `lockedUntil`, the function will return the value before executing the subsequent line:

```
block.timestamp > lockedPosition.lockedUntil
```

Location of Affected Code

File: veGUAN.sol#L158

```
function getVotingPowerOf(uint256 tokenId) public view returns (uint256
    scalingFactor, uint256 votingPower) {
  // load veGUAN's storage pointer
  VeGuanStorage storage $ = _getVeGuanStorage();
  // load the locked position's storage pointer
  LockedPositionData storage lockedPosition = _getVeGuanStorage().
     lockedPositions[tokenId];

  //if position is expired, return zero rather than reverting, will
     return zero in case of flashloan attack.
  if(lockedPosition.lockedUntil <= block.timestamp) return (scalingFactor
     , votingPower);

  // calculate how many seconds are left until the position is unlocked,
     used to determine the voting power
  // note: if the position is unlocked, the following value is set as 0
     which returns a voting power of 0
  uint256 remainingLockDuration =
    block.timestamp > lockedPosition.lockedUntil ? 0 : lockedPosition.
       lockedUntil - block.timestamp; //@audit change to  lockedPosition.
       lockedUntil - block.timestamp

  return _calculateVotingPower(
    ud60x18($.votingPowerCurveAFactor), ud60x18(remainingLockDuration),
       ud60x18(lockedPosition.stake)
  );
}
```

Recommendation

Consider applying the following changes:

```
function getVotingPowerOf(uint256 tokenId) public view returns (uint256
    scalingFactor, uint256 votingPower) {
  // code

-   uint256 remainingLockDuration = block.timestamp > lockedPosition.
    lockedUntil ? 0 : lockedPosition.lockedUntil - block.timestamp;
+   uint256 remainingLockDuration = lockedPosition.lockedUntil - block.
    timestamp;

  return _calculateVotingPower(
    ud60x18($.votingPowerCurveAFactor), ud60x18(remainingLockDuration),
       ud60x18(lockedPosition.stake)
  );
}
```

Team Response

Fixed.

[I-02] Current Logic In `_calculateVotingPower()` Will Add `1e18` to the `scalingFactorX18`

Severity

Informational

Description

In the current implementation of the `_calculateVotingPower()` function, the logic always adds 1e18 to the `scalingFactorX18` result, this function should be checked by the developers to make sure this is intended behavior.

Location of Affected Code

File: veGUAN.sol#L369

```
function _calculateVotingPower(
  UD60x18 votingPowerCurveAFactorX18,
  UD60x18 remainingLockDurationX18,
  UD60x18 positionStakeX18
)
  internal
  pure
  returns (uint256 scalingFactor, uint256 votingPower)
{
  // calculate the lock multiplier as explained in the function's natspec
  UD60x18 scalingFactorX18 = votingPowerCurveAFactorX18.mul(
     remainingLockDurationX18).add(UNIT); //@audit we add 1e18 to the
     value

  // return the scaling factor and voting power\
  scalingFactor = scalingFactorX18.intoUint256();
  votingPower = positionStakeX18.mul(scalingFactorX18).intoUint256();
}
```

Impact

Adding 1e18 to the \codex{scalingFactorX18‘ can affect the voting power.

Recommendation

Consider removing the UINT if this is not in the logic design, otherwise, the code works as expected.

Team Response

Fixed.

[I-03] Unnecessary Extra Call to Get Storage Variable Value

Severity

Informational

Description

An unnecessary call to get the storage variable value is being made in the function `getVotingPowerOf()`
.

Location of Affected Code

File: src/veGUAN.sol#L153

```
VeGuanStorage storage $ = _getVeGuanStorage();
// load the locked position's storage pointer
LockedPositionData storage lockedPosition = _getVeGuanStorage().
    lockedPositions[tokenId];
```

Impact

Extra calls add to gas consumption.

Recommendation

Change the code to be similar to other functions in the contract such as `previewNewVotingPower()`
:

```
VeGuanStorage storage $ = _getVeGuanStorage();
LockedPositionData storage lockedPosition = $.lockedPositions[tokenId];
```

Team Response

Fixed.

[I-04] Current Key Hash for VRF Is Not Correct

Severity

Informational

Description

The current key hash that is set in the wheel contract is not correct, it points to Sepolia which should point to the base mainnet.

Current key hash:

```
bytes32 public constant VRF_KEY_HASH = 0
    x787d74caea10b2b357790d5b5247c2f63d1d91572a9846f780606e4d953677ae; //
    @audit this is for sepolia
```

the max fee set to 300k, these values can be less when the code is deployed on the base blockchain(the blockchain guan ecosystem planned to deploy their codebase on)

correct key hash:

```
0xdc2f87677b01473c763cb0aee938ed3341512f6057324a584e5944e786144d70
```

Location of Affected Code

File: src/games/WheelOfGuantune.sol#L93

```
bytes32 public constant VRF_KEY_HASH = 0
    x787d74caea10b2b357790d5b5247c2f63d1d91572a9846f780606e4d953677ae; //
    @audit this is for sepolia
```

Impact

Incorrect set of the VRF key hash.

Recommendation

Change the key hash to:

```
bytes32 public constant VRF_KEY_HASH = 0
    xdc2f87677b01473c763cb0aee938ed3341512f6057324a584e5944e786144d70; //
    30 gwei on base
```

Team Response

Fixed.

[I-05] User Can Emit Unstake Event Without Unstaking Any Amount

Severity

Informational

Description

The function `unstake()` does not check if the current input amount is not zero, this will allow the user to emit `unstake` event without `unstake` any amount of token.

Location of Affected Code

File: src/veGUAN.sol#L327

```
function unstake(uint256 tokenId, uint256 amount) external onlyTokenOwner
    (tokenId) {
  // load veGUAN storage slot
  VeGuanStorage storage $ = _getVeGuanStorage();

  // load the lock data storage pointer
  LockedPositionData storage lockedPosition = $.lockedPositions[tokenId];

  // revert if the position is still locked
  if (block.timestamp < lockedPosition.lockedUntil) {
      revert PositionIsLocked();
  }

  // deduct the unstake amount from the locked position's state, if there
      isn't enough stake in the position the
  // call will revert with an underflow
  lockedPosition.stake -= amount;

  // transfer the lp tokens to the `msg.sender`
  IERC20($.lpToken).safeTransfer(msg.sender, amount);

  // cache the veGUAN's voting power
  (, uint256 votingPower) = getVotingPowerOf(tokenId);

  // emit an event
  emit LogUnstake(msg.sender, tokenId, lockedPosition.stake, votingPower)
      ;
}
```

Impact

Users can emit `unstake` event without unstaking any token amount.

Recommendation

Check for `amount > 0` to prevent emitting without unstaking.

Team Response

Fixed.

[I-06] No Need to Do State Updates When The `increaseAndStake()` Function Called with Zero Stake Amount Update

Severity

Informational

Description

The current implementation of the `increaseStakeAndLock()` function allows users to update their stake amount or lock duration. When the lock duration is updated, there's no need to modify the storage for the stake amount. This approach can help users save on gas fees.

Location of Affected Code

File: src/veGUAN.sol#L283

Impact

Updating the stake storage is unnecessary when no additional amount is added.

Recommendation

The current implementation can be optimized by adopting the following approach:

```
if (stakeIncrease > 0) {
  uint256 newStakeValue = lockedPosition.stake + stakeIncrease;

  // updates the locked position data
  lockedPosition.stake = newStakeValue;
  lockedPosition.lockedUntil = newUnlockTimestamp;
  // finally, transfer the lp tokens from the `msg.sender`
  IERC20($.lpToken).safeTransferFrom(msg.sender, address(this),
     stakeIncrease);
}
```

Team Response

Acknowledged.

shieldify

Thank you!