Guanciale AI DAO Security Review

Reviewed by: 0K_sec (aka zeroK)

Contents

1. About 0K security (AKA zeroK)

0K security known as zeroK is researcher with two years of experience in the in the Web3 security field. Specializing in Solidity, Sway, EVMs, FuelVM, and DeFi protocols. Starting as a warden on CodeArena then transitioned to bug hunting and ranked as elite on Immunefi.

2. Disclaimer

The details shared in this report are for informational purposes only and are not in intended to encourage or discourage users or investors from engaging with the mentioned bug bounty program. This report highlights a vulnerabilities i discovered while reviewing the specified protocol during a set period in specifi time and repository, Please conduct your own research and due diligence before investing in or working on mentioned protocol.

3. About Guanciale AI DAO

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.

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 - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

4.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5. Security Assessment Summary

The security review was carried out by the Shieldify team over six days of working. examined the veGuan and WheelOfGuantune contracts, as well as other related contracts. By the end of the review, I had identified 13 vulnerabilities: 2 critical, 1 high, 3 medium, and 7 low or informational issues.

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. Finding

6.1. Critical Findings

[C-01] 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

_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 Proof of Concept

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

6.2. High Findings

[H-01] 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 transaction will be executed without re-verting 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 docu-mentation states that proposals are reviewed by at proposals are reviewed by the council, which may prevent this issue fro from being executed

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.

6.3. Medium Findings

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

Severity

Medium Risk

Description

The function increaseStakeAndLock() is meant to allow users to increase stak amount AND/OR increase their lock durations, however, 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 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 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 token 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 cur-rent 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:

```
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 )
{
```

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() func-tion.

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.

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] 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 re-quires trusting them not to make malicious updates. One potential risk is freezing funds in the con-tract 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.

Impact

Should the LP token address be changed any user unstaking would potentially receive a totally dif-ferent 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.

6.4. Low Findings

[L-01] 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

```
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:

```
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.

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-02] 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.

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.

6.5. Informational Findings

[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
```

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.

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 ex-pected.

Team Response

Fixed.

[I-03] 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:

```
0 xdc2f87677b01473c763cb0aee938ed3341512f6057324a584e5944e786144d70
```

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-04] 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.

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-05] 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 stor-age for the stake amount. This approach can help users save on gas fees.

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.