OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 28	2022-01-11	Zapmore, DoD4uFN	Audit Final

Audit Notes

Audit Date	2021-11-22 - 2022-01-11
Auditor/Auditors	DoD4uFN, thing_theory
Auditor/Auditors Contact Information	contact@obeliskauditing.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB588569869

Disclaimer

This audit is not financial, investment, or any other kind of advice and is for informational purposes only. This report is not a substitute for doing your own research and due diligence. Obelisk is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Obelisk has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Obelisk is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. The audit is paid by the project but neither the auditors nor Obelisk has any other connection to the project and has no obligations other than to publish an objective report. Obelisk will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Obelisk assumes that the provided information and material were not altered, suppressed, or misleading. This report is published by Obelisk, and Obelisk has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Obelisk.

Obelisk Auditing

Defi is a relatively new concept but has seen exponential growth to a point where there is a multitude of new projects created every day. In a fast-paced world like this, there will also be an enormous amount of scams. The scams have become so elaborate that it's hard for the common investor to trust a project, even though it could be legit. We saw a need for creating high-quality audits at a fast phase to keep up with the constantly expanding market. With the Obelisk stamp of approval, a legitimate project can easily grow its user base exponentially in a world where trust means everything. Obelisk Auditing consists of a group of security experts that specialize in security and structural operations, with previous work experience from among other things, PricewaterhouseCoopers. All our audits will always be conducted by at least two independent auditors for maximum security and professionalism.

As a comprehensive security firm, Obelisk provides all kinds of audits and project assistance.

Audit Information

The auditors always conducted a manual visual inspection of the code to find security flaws that automatic tests would not find. Comprehensive tests are also conducted in a specific test environment that utilizes exact copies of the published contract.

While conducting the audit, the Obelisk security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Obelisk assesses the risks and assigns a risk level to each section together with an explanatory comment. Take note that the comments from the project team are their opinion and not the opinion of Obelisk.

Table of Contents

Project Information

Name	Gembites
Description	The most fair, transparent, community-owned decentralized casino in the world.
Website	https://gembites.com/
Contact	@KitsTelegram
Contact information	@KitsTelegram on TG
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Polygon

Audit of Gembites Keno

> **All findings of severity were either solved or commented on by the Gembites dev team.**

Obelisk was commissioned by Gembites on the 22nd of November 2021 to conduct a comprehensive audit of Gembites' Keno contracts. The following audit was conducted between the 22nd of November 2021 and the 11th of January 2021. Two of Obelisk's security experts went through the related contracts manually using industry standards to find if any vulnerabilities could be exploited either by the project team or users.

During the audit of the Gembites Keno contracts, there were findings of different severity. While presenting these findings to the project team, they worked to either solve the issues or comment on them. The most severe issue was issue #1 which was completely solved.

Issue #2 and issue #12 is still present. Issue #2 was commented on by the project team for which the user can create their own idea of the issue. Issue #12 is regarding a timelock that is currently not present.

The informational findings are good to know while interacting with the project but don't directly damage the project in its current state, hence it's up to the project team if they deem that it's worth solving these issues.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the Gembites project**.**

Please read the full document for a complete understanding of the audit.

Summary Table

Finding	ID	Severity	Status
Users Can Bet Multiple Times Same Number On A Single Bet	#0001	High Risk	Closed
Calling FulfillRandomness Can Revert	#0002	Medium Risk	Open
Unbounded Loop	#0003	Low Risk	Closed
Block Confirmation Time	#0004	Low Risk	Mitigated
No Limit For Protocol Values	#0005	Informational	Closed
Redundant Assignment	#0006	Informational	Mitigated
Bet Info Cannot Be Tracked	#0007	Informational	Open
Missing Zero Checks	#0008	Informational	Mitigated
ChangeBetAmount Can Be Front-Run	#0009	Medium Risk	Closed
No Events Emitted For Changes To Protocol Values	#00010	Informational	Closed
Compilation Errors	#00011	Informational	Closed
No Timelock	#0012	Low Risk	Open
Unverified Token Contract	#00013	Informational	Open

Findings

Manual Analysis

Users Can Bet Multiple Times Same Number On A Single Bet

FINDING ID	#0001
SEVERITY	High Risk
STATUS	Closed
LOCATION	Keno.sol -> 95-117

```
function bet(uint256[] memory _numbers) external nonReentrant {
    require(
        _numbers.length < 8 && _numbers.length > 0,
        "Keno: Every ticket should have 1 to 7 numbers."
    );

    GBTS.safeTransferFrom(msg.sender, address(ULP), betAmount);

    totalBettedAmount += betAmount;

    bytes32 requestId = RNG.requestRandomNumber();
    bool[51] memory gameNumbers;

    requestToBet[requestId] = BetInfo(
        msg.sender,
        requestId,
        _numbers.length,
        _numbers,
        gameNumbers
    );

    emit BetStarted(requestToBet[requestId]);
}
```

DESCRIPTION	The *bet()* function does not check the *numbers* array for duplicate numbers. The users are able to bet the same number multiple times on a single bet. E.g. Bet: [9,9,9,9,9,9,9], Drawn Numbers: [9,11,17,43,49,32,12] In such a situation the user wins the maximum multiplier by matching only a single number.

RECOMMENDATION	In the *bet()* function, do not allow duplicate values in the *_numbers* array.
RESOLUTION	The project team has implemented the recommended fix.

Calling FulfillRandomness Can Revert

FINDING ID	#0002
SEVERITY	Medium Risk
STATUS	Open
LOCATION	RandomNumberGenerator.sol -> 88-97

```
1    function fulfillRandomness(bytes32 _requestId, uint256
  _randomness)
2        internal
3        override
4    {
5        currentRandomNumber = _randomness;
6        IGame GAME = IGame(requestToGame[_requestId]);
7        GAME.play(_requestId, _randomness);
8
9        emit randomNumberArrived(true, _randomness, _requestId);
10   }
```

LOCATION	Keno.sol -> 156-159

```
1        if (amountToSend > 0) {
2            totalWinnings += amountToSend;
3            ULP.sendPrize(msg.sender, amountToSend);
4        }
```

DESCRIPTION	Calls to *fulfillRandomness()* can revert due to the indeterminate gas cost of the unbound loop in the *play()* function. The chainlink docs state that FulfillRandomness must not revert. FulfillRandomness may also revert if the call to *ULP* on line 158 reverts, perhaps due to insufficient balance.
RECOMMENDATION	Instead of transferring funds within the *play()* function, make the winnings claimable by the user.
RESOLUTION	Team comment: *While it is technically possible for the ULP to run out of GBTS,*

it's a similar situation to a casino where we rely on the law of large numbers to prevent that from happening.
Running out of GBTS in the ULP is exceedingly unlikely, and in the event, it did happen, whether or not the VRF reverted wouldn't matter.

Unbounded Loop

FINDING ID	#0003
SEVERITY	Low Risk
STATUS	Closed
LOCATION	Keno.sol -> 131-142

```
1         while (size < 15) {
2             uint256 gameNumber = (uint256(
3                 keccak256(abi.encode(_randomNumber,
  address(msg.sender), nonce))
4             ) % 50) + 1;
5
6             nonce++;
7
8             if (!betInfo.gameNumbers[gameNumber]) {
9                 betInfo.gameNumbers[gameNumber] = true;
10                size++;
11            }
12        }
```

DESCRIPTION	This loop attempts to select 15 random numbers from a set of 51 possible choices. This is vulnerable to collision and can cause the loop to iterate an indeterminate number of times. IIUC the formula *1 - ((C-1)/C)^(N(N-1)/2)* where C is the number of possible choices and N is the number of selection, describes the approximate change of collision (simplified birthday problem). Thus *1 - ((51-1)/51)^(15(15-1)/2) = 0.8749792888* meaning there is approximately an 87% chance of collision.
RECOMMENDATION	Create an array with the possible values, select a random number within the current array length, pop the selected item, remove it from the array, and resize the length. [0,1,2,3,4,5] -> pick index 3 -> [0,1,2,5,4,5] -> [0,1,2,5,4]
RESOLUTION	The project team has implemented the recommended fix.

Block Confirmation Time

FINDING ID	#0004
SEVERITY	Low Risk
STATUS	Mitigated
LOCATION	RandomNumberGenerator.sol

DESCRIPTION	Choosing a safe block confirmation time is important for the integrity of the platform. Chainlink docs - Choose a safe block confirmation time Polygon's VRF response time is 10 blocks by default. In contrast, Binance has a confirmation time of 128 blocks for deposits.
RECOMMENDATION	Decide how many confirmations blocks your VRF should have.
RESOLUTION	The project team commented: *They have determined a 10 block response time is long enough, and it's also a balance between security and responsiveness.*

No Limit For Protocol Values

FINDING ID	#0005
SEVERITY	Informational
STATUS	Closed
LOCATION	Keno.sol -> 170-174

```
1     function changeBetAmount(uint256 _newBetAmount) external
  onlyOwner {
2         betAmount = _newBetAmount;
3
4         emit BetAmountChanged(_newBetAmount);
5     }
```

DESCRIPTION	The following values can be set arbitrarily high, potentially breaking the functionality of the contracts: • *betAmount*
RECOMMENDATION	Add an upper limit to the values e.g. less than the supply.
RESOLUTION	The project team has implemented the recommended fix.

Redundant Assignment

FINDING ID	#0006
SEVERITY	Informational
STATUS	Mitigated
LOCATION	• Keno.sol -> 161: *betInfo.gameNumbers[0] = true;*

DESCRIPTION	The aforementioned line doesn't have an effect on the contract's functionality.
RECOMMENDATION	Remove any unnecessary assignments.
RESOLUTION	The project team commented: *Apparently, it's used to easily allow a UI to see if the bet is over or not without necessarily having to query events if the requestID is already known in the UI*.

Bet Info Cannot Be Tracked

FINDING ID	#0007
SEVERITY	Informational
STATUS	Open
LOCATION	• Keno.sol -> 52: *mapping(bytes32 => BetInfo) public requestToBet;*

DESCRIPTION	Since the mapping is using the *requestId* provided by the VRF Coordinator, there is no way to track the bets.
RECOMMENDATION	Add an array of all *requestId*.
RESOLUTION	The project team commented: *I think it is important to note that the events which include the requestIds are emitted when a bet is created and when the play function is called* *We can query and store those events in our backend, as they are also permanently stored on full nodes.* Not resolved, because there is no easy way for the users to track BetInfo. Adding a public array of all *requestId* would resolve the issue.

ChangeBetAmount Can Be Front-Run

FINDING ID	#0009
SEVERITY	Medium Risk
STATUS	Closed
LOCATION	*Revision 2* Keno.sol -> 133-144

```
totalBettedAmount += betAmount;

bytes32 requestId = RNG.requestRandomNumber();
bool[51] memory gameNumbers;

requestToBet[requestId] = BetInfo(
    msg.sender,
    requestId,
    _numbers.length,
    _numbers,
    gameNumbers
);
```

LOCATION	*Revision 2* Keno.sol -> 188-193

```
uint256 amountToSend = (multiplier * betAmount) / 100;

if (amountToSend > 0) {
    totalWinnings += amountToSend;
    ULP.sendPrize(msg.sender, amountToSend);
}
```

DESCRIPTION	Since *betAmount* is only tracked globally and can be updated, it is possible for a user to receive more winnings than intended based on the amount they bet. By placing a bet before the *betAmount* is increased, the user can receive additional winnings if their bet is executed after the *betAmount* is increased.

	Example: TX0 userA->bet() betAmount == 100 TX1 owner->changeBetAmount(150) betAmount == 150 TX2 RandomNumberGenerator->play(userA) userA receives multiplier * betAmount(150 instead of 100)
RECOMMENDATION	Add an additional field to the *BetInfo* struct, called *betAmount* which is set to the value of the contract variable *betAmount* at the time *bet()* function is called, and calculate the *amountToSend* using this struct member.
RESOLUTION	The project team has implemented the recommended fix.

Static Analysis

Missing Zero Checks

FINDING ID	#0008
SEVERITY	Low Risk
STATUS	Mitigated
LOCATION	RandomNumberGenerator.sol -> 47-64

```
1     constructor(
2         address _vrfCoordinator,
3         address _link,
4         bytes32 _keyHash,
5         uint256 _fee,
6         IUnifiedLiquidityPool _ULP
7     )
8         VRFConsumerBase(
9             _vrfCoordinator, // VRF Coordinator
10            _link // LINK Token
11        )
12    {
13        keyHash = _keyHash;
14        fee = _fee;
15        ULP = _ULP;
16
17        emit RandomNumberGeneratorDeployed();
18    }
```

LOCATION	Keno.sol -> 68-89

```
    constructor(
        IUnifiedLiquidityPool _ULP,
        IERC20 _GBTS,
        IRandomNumberGenerator _RNG
    ) {
        ULP = _ULP;
        GBTS = _GBTS;
        RNG = _RNG;

        betAmount = 100 * 10**18;

        winningTable.push();
        winningTable.push([0, 220]); // 0, 2.2
        winningTable.push([0, 0, 1030]); // 0, 0, 10.3
        winningTable.push([0, 0, 100, 2450]); //  0, 0, 1.00, 24.50
        winningTable.push([0, 0, 0, 700, 6900]); // 0, 0, 0, 7.00, 69.00
        winningTable.push([0, 0, 0, 100, 1600, 22900]); // 0, 0, 0, 1.00, 16.00, 229.00
        winningTable.push([0, 0, 0, 100, 200, 3900, 55400]); // 0, 0, 0, 1.00, 2.00, 39.00, 554.00
        winningTable.push([0, 0, 0, 0, 100, 1400, 22100, 199900]); //0, 0, 0, 0, 1.00, 14.00, 221.00, 1999.00

        emit KenoDeployed();
    }
```

DESCRIPTION	The *_newBetAmount* may be set to zero which will result in a winning bet winning 0 rewards and only cost gas. The *_ULP*, *_GBTS,* and *_RNG* addresses may all be zero, making the contract nonfunctional. The contract address values can be set to zero address in various constructors, initializers, and setter functions. Zero addresses may cause incorrect contract behavior.
RECOMMENDATION	Add a check to ensure contract values are never set to an invalid zero address.
RESOLUTION	*Keno.sol* has been resolved. *RandomNumberGenerator.sol* has not, the audit team will ensure the contract is deployed with correct values during on-chain analysis.

No Events Emitted For Changes To Protocol Values

FINDING ID	#00010
SEVERITY	Informational
STATUS	Closed
LOCATION	*Revision 2* Keno.sol -> 217-219

```
1     function changeBetsAllowed(bool _betsAllowed) external onlyOwner
  {
2         betsAllowed = _betsAllowed;
3     }
```

DESCRIPTION	Functions that change important variables should emit events such that users can more easily monitor the change.
RECOMMENDATION	Emit events from these functions.
RESOLUTION	The project team has implemented the recommended fix.

Compilation Errors

FINDING ID	#00011
SEVERITY	Informational
STATUS	Closed
LOCATION	*Revision 2* • Keno.sol -> 45: *bool public betsAllowed;* • Keno.sol -> 65-71: *modifier betsAllowed()* *Revision 2* Keno.sol -> 84-86

```
1         assert(address(_ULP) != address(0), "Keno: ULP cannot be 0
  address");
2         assert(address(_GBTS) != address(0), "Keno: GBTS cannot be 0
  address");
3         assert(address(_RNG) != address(0), "Keno: RNG cannot be 0
  address");
```

DESCRIPTION	The name *betsAllowed* is used as a variable and as a modifier.
RECOMMENDATION	Change the name of either the variable or the modifier. Replace the *assert* keyword with *require*.
RESOLUTION	The project team has implemented the recommended fix.

On-Chain Analysis

No Timelock

FINDING ID	#0012
SEVERITY	Low Risk
STATUS	Open
LOCATION	Keno.sol

DESCRIPTION	The following contracts have not had their ownership transferred to a timelock contract yet: - Keno.sol The contract owner can change the bet amount with no delay.
RECOMMENDATION	Transfer ownership to a timelock contract.
RESOLUTION	N/A

Unverified Token Contract

FINDING ID	#0013
SEVERITY	Informational
STATUS	Open
LOCATION	GBTS

DESCRIPTION	In our previous Gembites audit the GBTS token contract was verified and its address was the same. The contract is now unverified on Polygonscan
RECOMMENDATION	Verify the contract
RESOLUTION	N/A

Appendix A - Reviewed Documents

Document	Address
interfaces/IGame.sol	0x30eE5c68B3d5fADAaFA293cC8E5d2D6651a3524e
interfaces/IRandomNumberGenerator.sol	0xe03e2ade6ba3eeb880353368ea9a0f665cc668e8
interfaces/IUnifiedLiquidityPool	0xe03e2ade6ba3eeb880353368ea9a0f665cc668e8
Keno.sol	0xe03e2ade6ba3eeb880353368ea9a0f665cc668e8
RandomNumberGenerator.sol	0x30eE5c68B3d5fADAaFA293cC8E5d2D6651a3524e

Revisions

Revision 1	Keno.zip

Imported Contracts

Chainlink	0.2.2
OpenZeppelin	4.4.0

Externally Owned Accounts

Owner	0xEBFb9973A0e711f84541724AF7BB1E8e197B0915

External Contracts

These contracts are not part of the audit scope.

GBTS	0xbe9512e2754cb938dd69Bbb96c8a09Cb28a02D6D
Unified Liquidity Pool	0xbD658acCb3364b292E2f7620F941d4662Fd25749

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause the loss of all Tokens / Funds.
Medium Risk	A vulnerability that can cause the loss of some Tokens / Funds.
Low Risk	A vulnerability which can cause the loss of protocol functionality.
Informational	Non-security issues such as functionality, style, and convention.

Appendix C - Finding Statuses

Closed	Contracts were modified to permanently resolve the finding.
Mitigated	The finding was resolved by other methods such as revoking contract ownership. The issue may require monitoring, for example in the case of a time lock.
Partially Closed	Contracts were updated to fix the issue in some parts of the code.
Partially Mitigated	Fixed by project specific methods which cannot be verified on chain. Examples include compounding at a given frequency.
Open	The finding was not addressed.

Appendix D - Audit Procedure

A typical Obelisk audit uses a combination of the three following methods:

Manual analysis consists of a direct inspection of the contracts to identify any security issues. Obelisk auditors use their experience in software development to spot vulnerabilities. Their familiarity with common contracts allows them to identify a wide range of issues in both forked contracts as well as original code.

Static analysis is software analysis of the contracts. Such analysis is called “static” as it examines the code outside of a runtime environment. Static analysis is a powerful tool used by auditors to identify subtle issues and to verify the results of manual analysis.

On-chain analysis is the audit of the contracts as they are deployed on the block-chain. This procedure verifies that:

- deployed contracts match those which were audited in manual/static analysis;
- contract values are set to reasonable values;
- contracts are connected so that interdependent contract function correctly;
- and the ability to modify contract values is restricted via a timelock or DAO mechanism. (We recommend a timelock value of at least 72 hours)

Each obelisk audit is performed by at least two independent auditors who perform their analysis separately.

After the analysis is complete, the auditors will make recommendations for each issue based on best practice and industry standards. The project team can then resolve the issues, and the auditors will verify that the issues have been resolved with no new issues introduced.

Our auditing method lays a particular focus on the following important concepts:

- Quality code and the use of best practices, industry standards, and thoroughly tested libraries.
- Testing the contract from different angles to ensure that it works under a multitude of circumstances.
- Referencing the contracts through databases of common security flaws.

Follow Obelisk Auditing for the Latest Information

 ObeliskOrg

 ObeliskOrg

OBELISK

Part of Tibereum Group