OBELISK

OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 48	2021-08-08	Zapmore, Donut	Audit Final

Audit Notes

Audit Date	2021-07-08 - 2021-08-07
Auditor/Auditors	Donut, Plemonade, DoD4uFN
Auditor/Auditors Contact Information	tibereum-obelisk@protonmail.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB586161658

Disclaimer

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.

Table of Content

Project Information

Project Name	Gembites
Description	One of the most unique decentralized casinos ever seen. Everything from house funds, to game contracts, are run on the blockchain, with no possibility of third parties skewing the odds.
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

Executive Summary

The audit of Gembites was conducted by three of Obelisks’ security experts between the 8th of July 2021 and the 7th of August 2021.

After finishing the full audit, Obelisk can safely state that there were some vulnerabilities that could cause issues to the project. Obelisk gave Gembites an opportunity to resolve these issues found, which they swiftly did. After a re-audit was done, all serious issues found had been mitigated or commented on.

The on-chain analysis currently ONLY refers to the core contracts of UnifiedLiquidityPool and random number generator.

Other Informational findings are there for informational purposes and don’t impact the project on a larger scale on audited implementation.

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

Audited Part	ID	Severity	Note
Game Randomness Can Be Frontrun	#0001	High Risk	Mitigated
Game Approval Can Be Toggled Freely After Timelock	#0002	Medium Risk	Mitigated
List Of Approved Games Not Easily Accessible	#0003	Medium Risk	Mitigated
Prizes Cannot Be Claimed When Games Are Paused Or Locked	#0004	Medium Risk	Mitigated
Prize Cannot Be Claimed If Balance Is Insufficient	#0005	Low Risk	See Comment
Lottery Purchases Paid To Lottery Contract	#0006	Low Risk	Mitigated
Incorrect Randomness Function Call	#0007	Low Risk	Mitigated
Profit Distribution Is Done One At A Time	#0008	Informational	See Comment
DiceRoll Minimum Bet Value	#0009	Informational	Mitigated
Address Of Zero Staker Can Be Modified	#0010	Informational	Mitigated
Ticket Limit Can Be Bypassed	#0011	Informational	See Comment
No Events Emitted For Changes To Protocol Values	#0012	Informational	Mitigated
Protocol Values Should Be Public	#0013	Informational	See Comment
Contract Variables Set But Never Used	#0014	Informational	Mitigated
Contract Variable Used But Never Set	#0015	Informational	Mitigated
Unused Events	#0016	Informational	Mitigated
Shares Only Deducted, Not Burned	#0017	Informational	Mitigated

Chainlink VRF Call Does Not Match Signature	#0018	Informational	Mitigated
Invalid Import Path	#0019	Informational	Mitigated
Contract Value Can Be Constant or Immutable	#0020	Informational	See Comment
Use Safe Transfer	#0021	Informational	Mitigated
Different Versions Of Solidity	#0022	Informational	Mitigated
Unbound Loop	#0023	Informational	Mitigated
RandomNumberConsumer can be disconnected from ULP	#0024	Medium Risk	Mitigated
Multiple Versions of OpenZeppelin	#0025	Informational	Mitigated

Introduction

Obelisk was commissioned by Gembites on the 2nd of July 2021 to conduct a comprehensive audit of Gembites’ contract. The following audit was conducted between the 8th of July 2021 and the 7th of August 2021 and delivered on the 8th of August 2021. Three of Obelisk's security experts went through the related contracts using industry standards to find if any vulnerabilities could be exploited.

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

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.

The audit was conducted on contracts that were not yet live in a production environment. A comprehensive on-chain analysis was conducted as the contracts went live in order to match the audited contracts with the published contracts. The on-chain analysis currently ONLY refers to the core contracts of *UnifiedLiquidityPool* and *RandomNumberGenerator* as the other contracts were not yet deployed.

At the first run-through of the audit, there were multiple findings of all risk severity levels that could cause a problem while using the contracts. Obelisk gave both the findings and recommended solutions to the findings to the Gembites team. The Gembites team worked to solve all severe findings and presented Obelisk with new contracts. These new contracts were then re-audited in order to make sure the findings were solved and no new vulnerabilities were introduced.

All severe findings were found to be solved and/or commented on and no new issues were introduced.

The informational findings are good to know while interacting with the project but don’t directly damage the project in its current state.

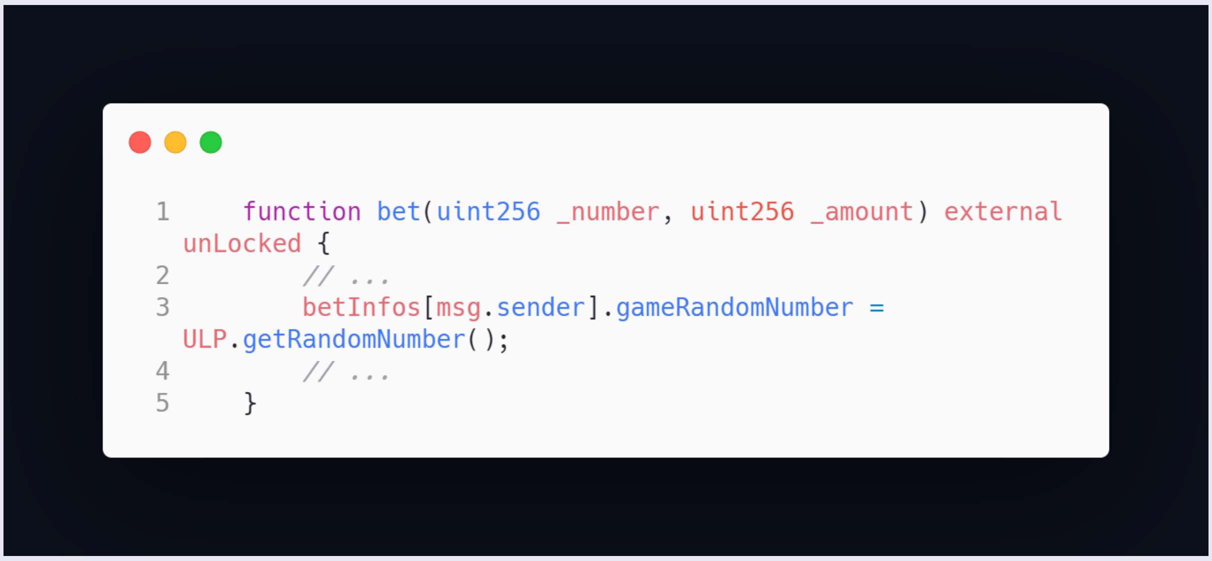
Please see each section of the audit to get a full understanding of the audit.

Findings

Manual Analysis

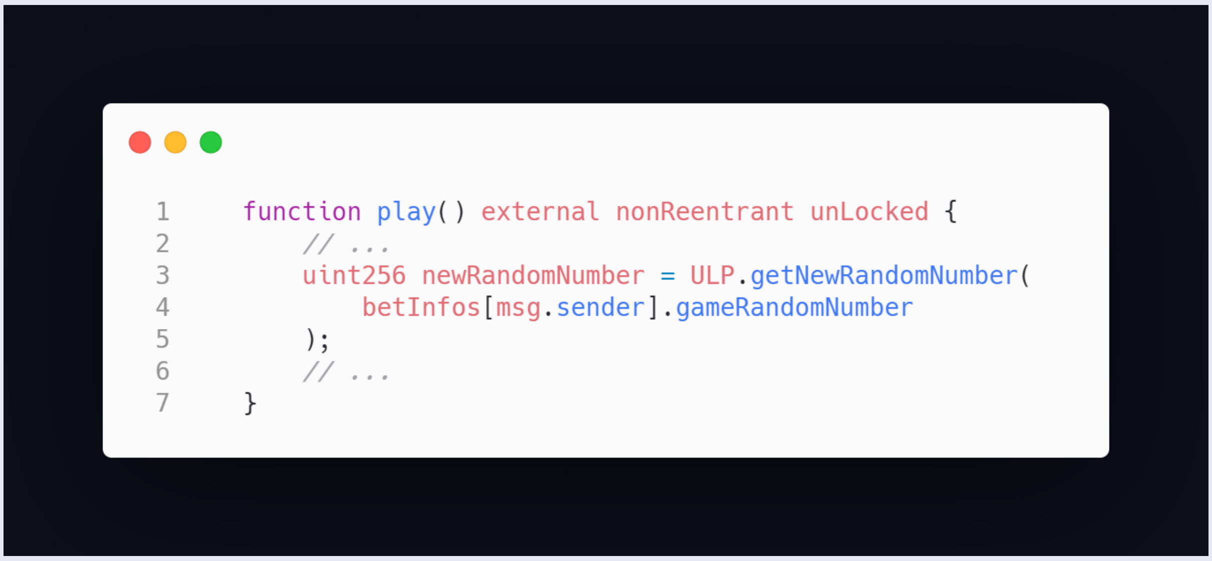
Game Randomness Can Be Frontrun

SEVERITY	High Risk
RESOLVED	**YES**
FINDING ID	#0001
LOCATION	DiceRoll.sol -> 86-119 RockPaperScissors.sol -> 80-113



```
function bet(uint256 _number, uint256 _amount) external unLocked {
    // ...
    betInfos[msg.sender].gameRandomNumber = ULP.getRandomNumber();
    // ...
}
```

LOCATION	DiceRoll.sol -> 124-157 RockPaperScissors.sol -> 118-163



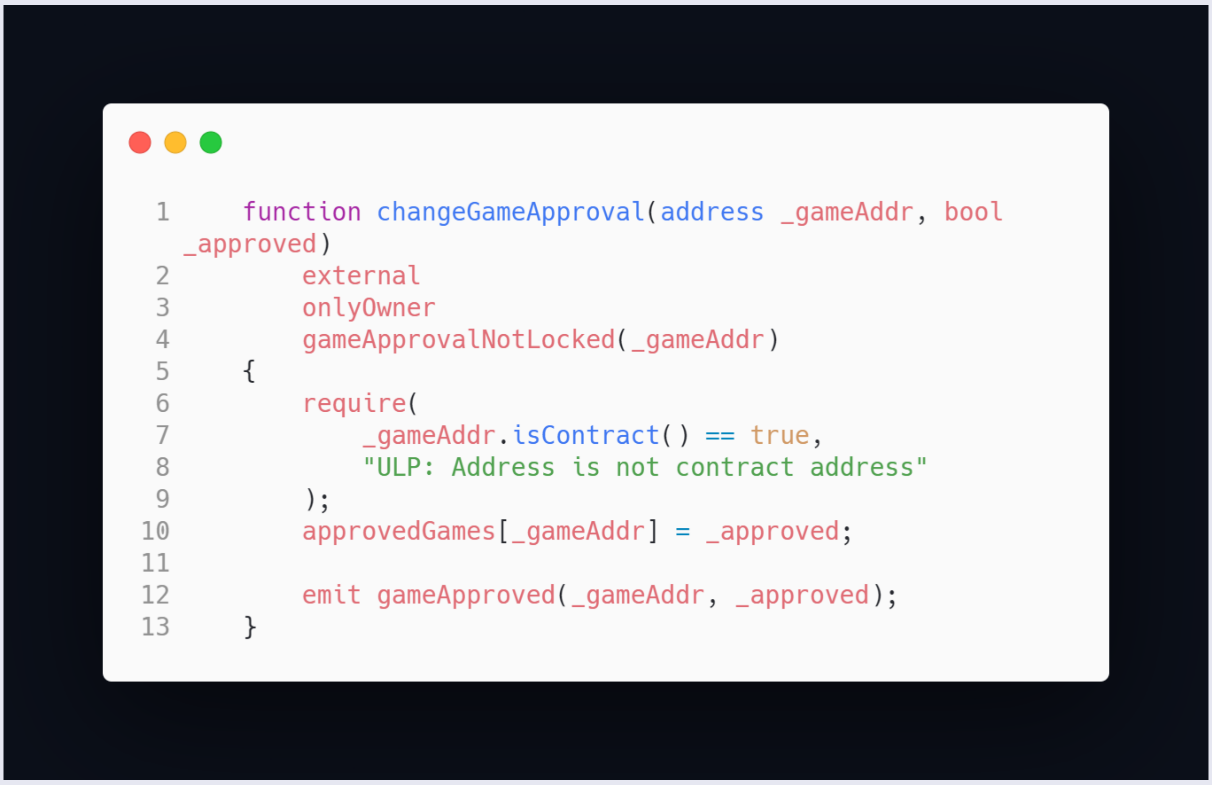
```
function play() external nonReentrant unLocked {
    // ...
    uint256 newRandomNumber = ULP.getNewRandomNumber(
        betInfos[msg.sender].gameRandomNumber
    );
    // ...
}
```

DESCRIPTION	The result of the VRF randomness call can be front-run. An

	actor watching for randomness contract transactions will be able to identify when a randomness request is being fulfilled and purchase a dice roll using a higher priority (gas fee). As a result, they will be able to guarantee their wins with no risk and maximum returns. Note that because all games share the same randomness mechanism, this vulnerability can express itself from the randomness requests of other games. As per Chainlink's documentation (https://docs.chain.link/docs/vrf-security-considerations/#dont-accept-bidsbetsinputs-after-you-have-made-a-randomness-request) , it is important to prevent inputs after making a randomness request.
RECOMMENDATION	Reserve unique randomness request ids per call to the randomness function. Prevent multiple games from interacting indirectly via the randomness function.
MITIGATED/COMMENT	The project team has implemented a system where randomness requests may be batched in groups of 1 to 4 blocks. As randomness requests are not returned until after 10 blocks on Polygon, this can be considered acceptable.

Game Approval Can Be Toggled Freely After Timelock

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0002
LOCATION	UnifiedLiquidityPool.sol -> 388-400

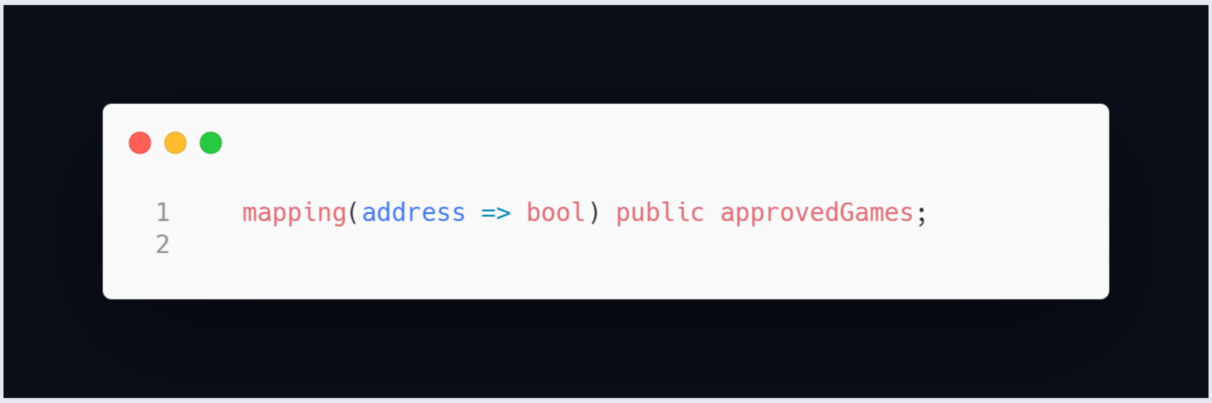


```
    function changeGameApproval(address _gameAddr, bool
_approved)
        external
        onlyOwner
        gameApprovalNotLocked(_gameAddr)
    {
        require(
            _gameAddr.isContract() == true,
            "ULP: Address is not contract address"
        );
        approvedGames[_gameAddr] = _approved;

        emit gameApproved(_gameAddr, _approved);
    }
```

DESCRIPTION	The approval timelock is only set once, and after it expires the contract owner may toggle the approval of a game freely. This can break the operation of games relying on the UnifiedLiquidityPool.
RECOMMENDATION	Reset the timelock every time the game approval changes, giving users time to respond to changes to the approval state.
MITIGATED/COMMENT	The timelock is reset after every change to a game's approval.

List Of Approved Games Not Easily Accessible

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0003
LOCATION	UnifiedLiquidityPool.sol -> 67

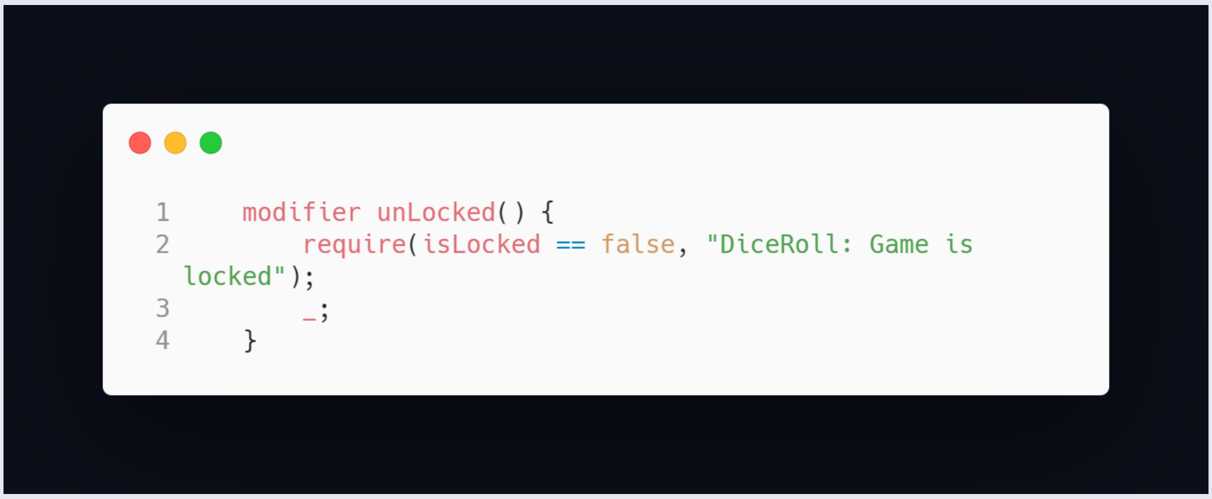


```
mapping(address => bool) public approvedGames;

```

DESCRIPTION	The approved games can only be queried on a per address basis. This makes it challenging to track which games are approved. Note: Approved games can call *sendPrize* and withdraw GBTS from the *UnifiedLiquidityPool*.
RECOMMENDATION	Add an array which lists all games which are approved or under timelock to be approved.
MITIGATED/COMMENT	Project team added a list which contains all approved game addresses.

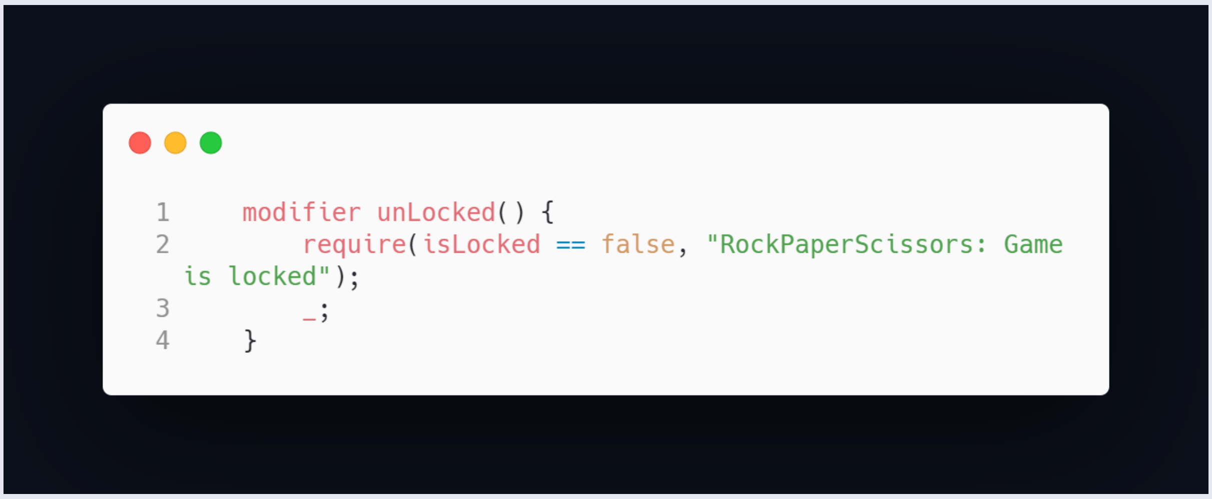
Prizes Cannot Be Claimed When Games Are Paused Or Locked

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0004
LOCATION	DiceRoll.sol -> 55-58



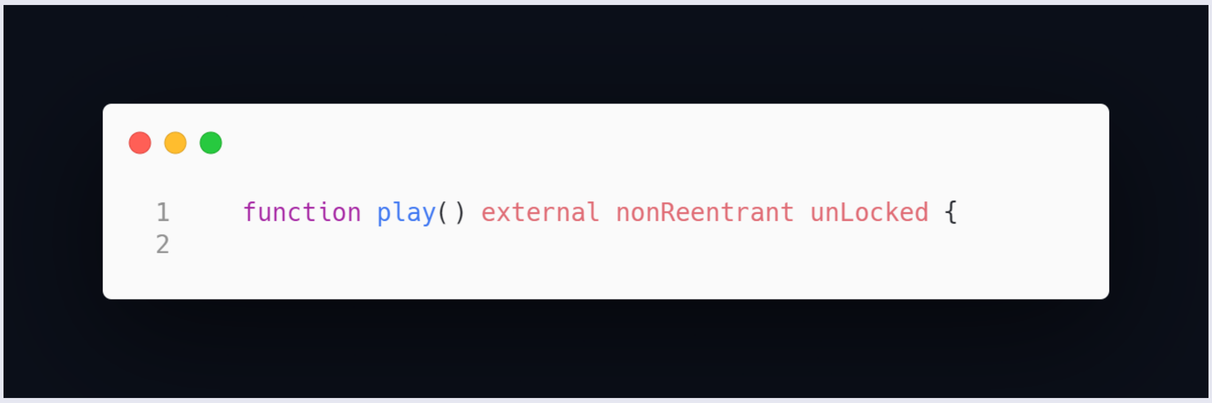
```
    modifier unLocked() {
        require(isLocked == false, "DiceRoll: Game is locked");
        _;
    }
```

LOCATION	RockPaperScissors.sol -> 49-52



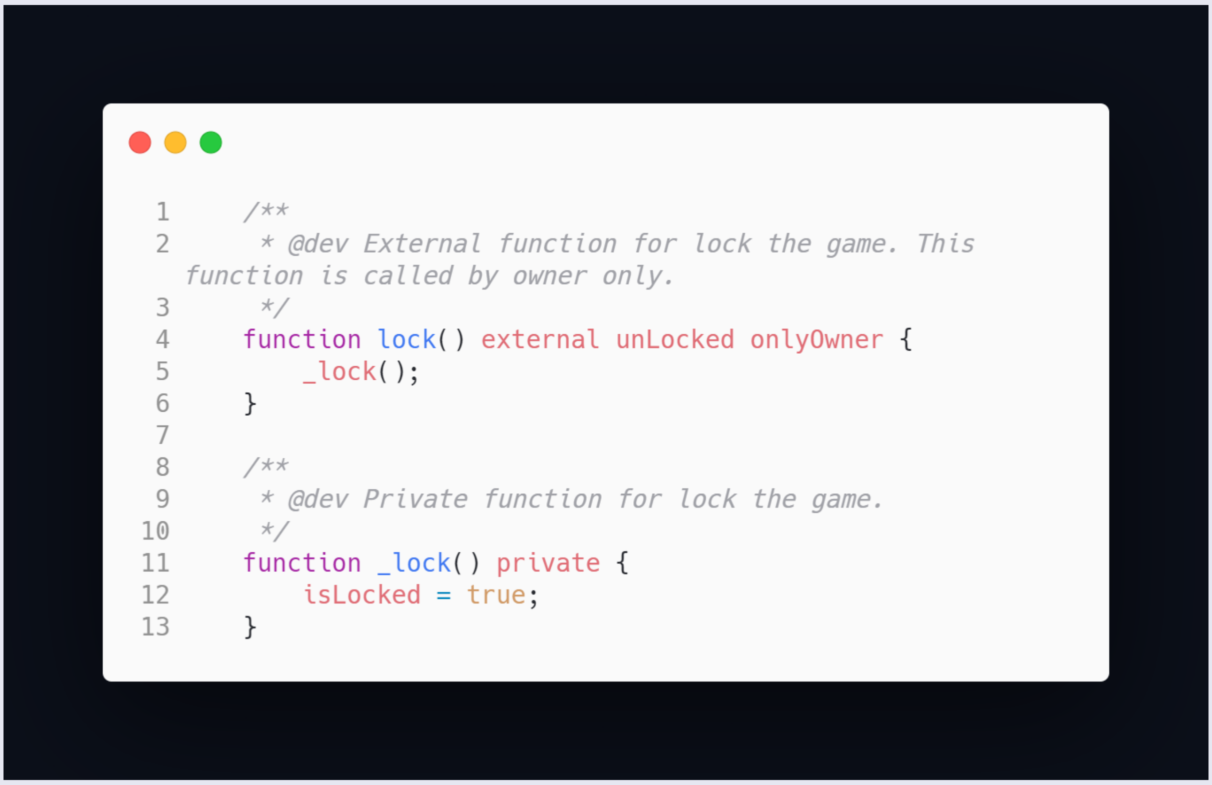
```
    modifier unLocked() {
        require(isLocked == false, "RockPaperScissors: Game is locked");
        _;
    }
```

LOCATION	DiceRoll.sol -> 124 RockPaperScissors.sol -> 118



```
function play() external nonReentrant unLocked {

```

LOCATION	DiceRoll.sol -> 186-198 RockPaperScissors.sol -> 192-204

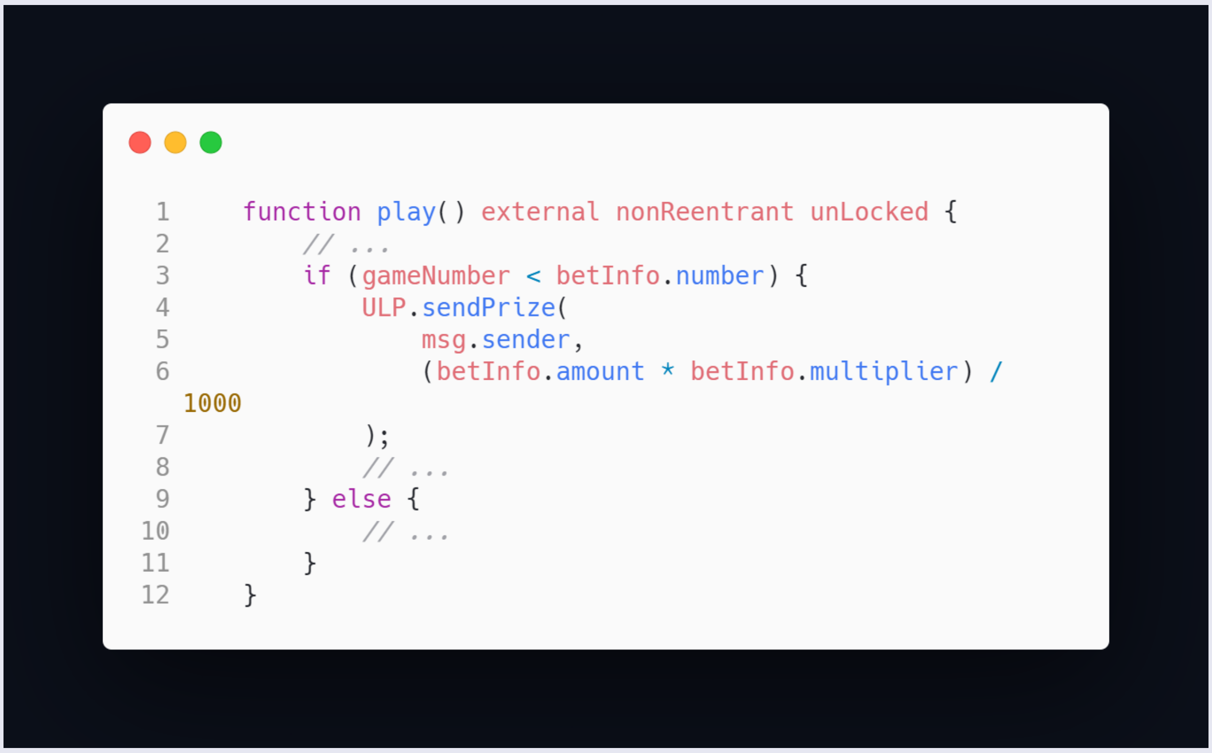


```
/**
 * @dev External function for lock the game. This function is called by owner only.
 */
function lock() external unLocked onlyOwner {
    _lock();
}

/**
 * @dev Private function for lock the game.
 */
function _lock() private {
    isLocked = true;
}
```

DESCRIPTION	The game can be paused, preventing users from receiving winnings.
RECOMMENDATION	Allow users to claim winnings when the game is locked.
MITIGATED/COMMENT	Lottery prizes can be claimed when the game is paused. DiceRoll and RockPaperScissors no longer have a pause mechanism.

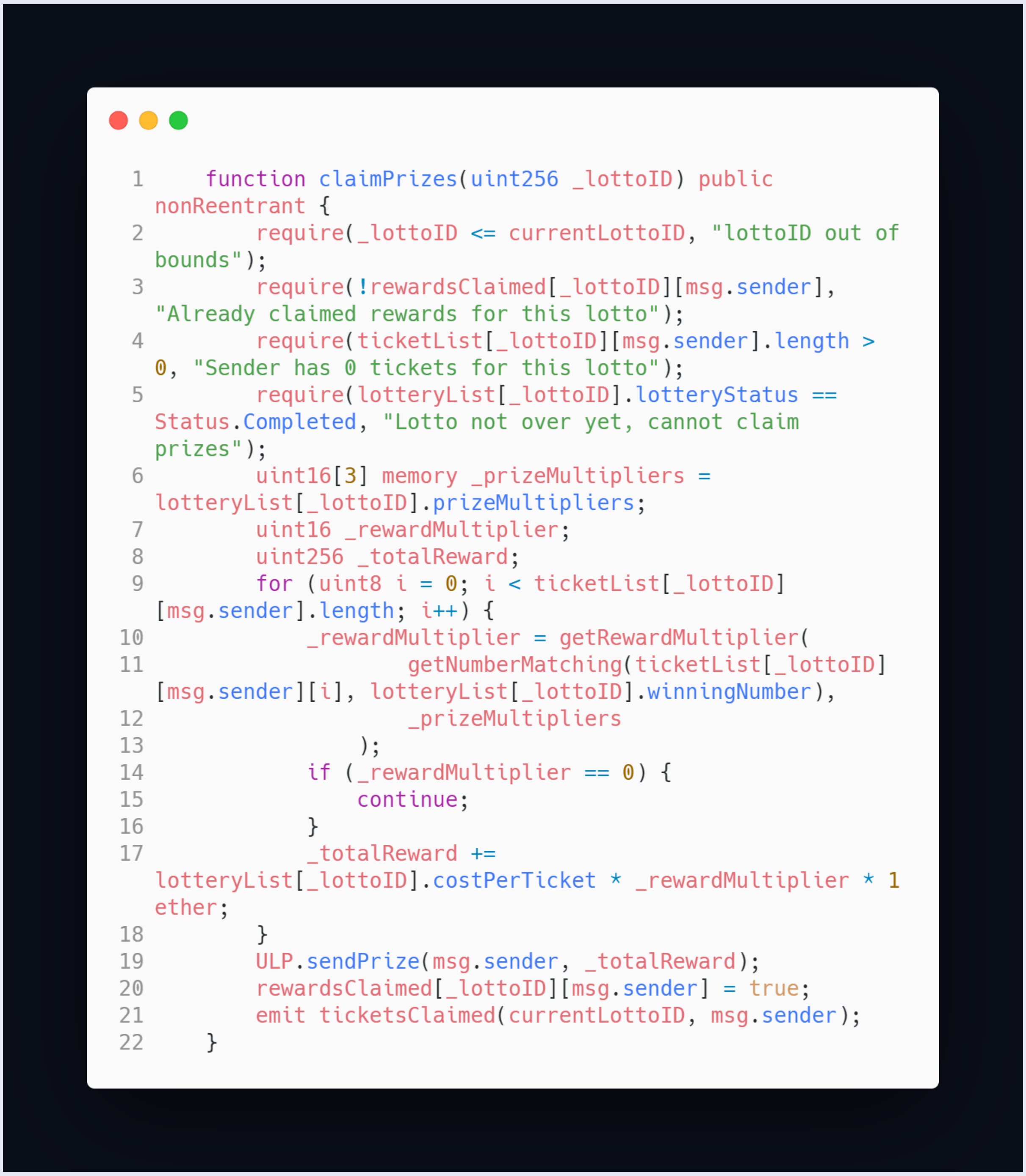
Prize Cannot Be Claimed If Balance Is Insufficient

SEVERITY	Low Risk
RESOLVED	**NO**
FINDING ID	#0005
LOCATION	DiceRoll.sol -> 124-157



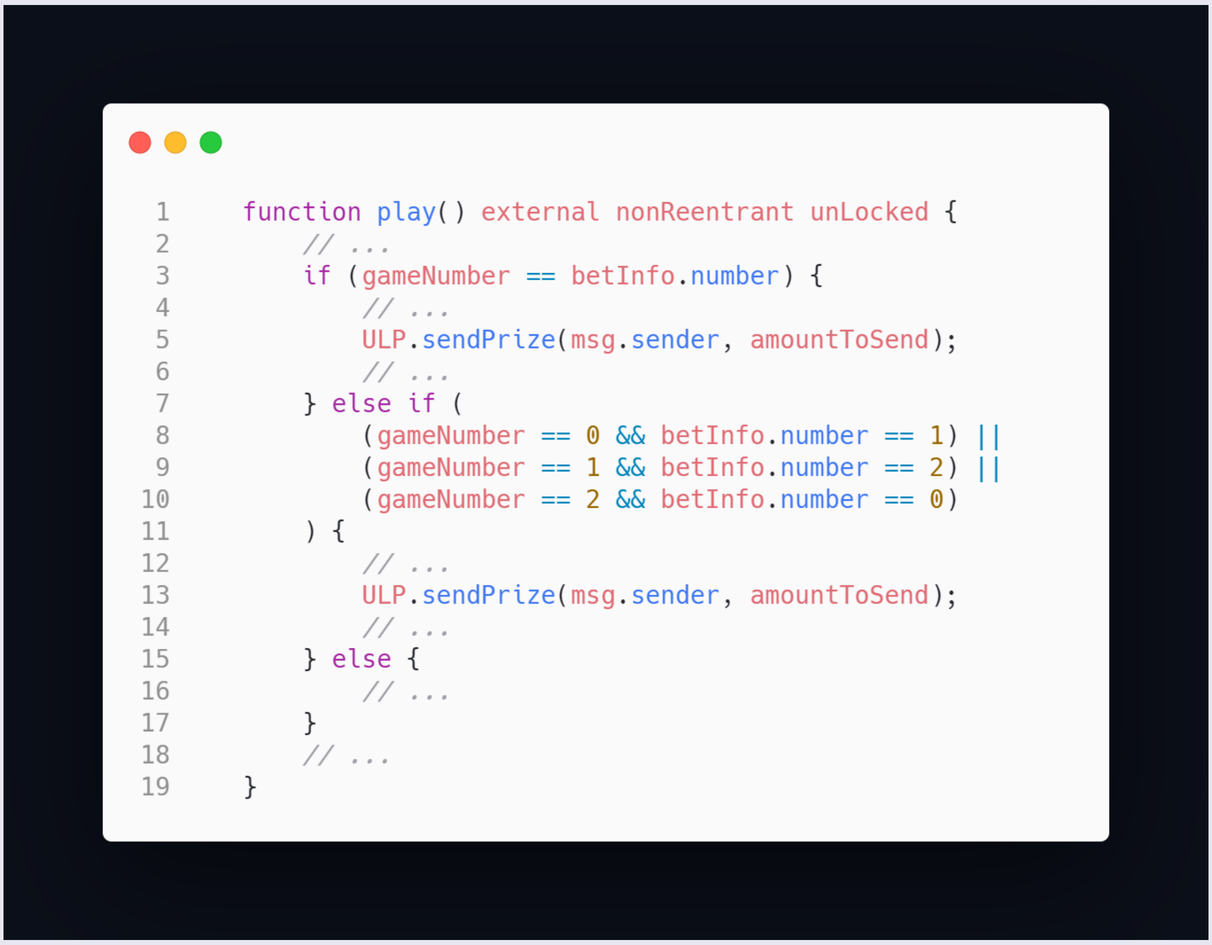
```
function play() external nonReentrant unLocked {
    // ...
    if (gameNumber < betInfo.number) {
        ULP.sendPrize(
            msg.sender,
            (betInfo.amount * betInfo.multiplier) /
1000
        );
        // ...
    } else {
        // ...
    }
}
```

LOCATION	Lottery.sol -> 124-145



```
    function claimPrizes(uint256 _lottoID) public nonReentrant {
        require(_lottoID <= currentLottoID, "lottoID out of bounds");
        require(!rewardsClaimed[_lottoID][msg.sender], "Already claimed rewards for this lotto");
        require(ticketList[_lottoID][msg.sender].length > 0, "Sender has 0 tickets for this lotto");
        require(lotteryList[_lottoID].lotteryStatus == Status.Completed, "Lotto not over yet, cannot claim prizes");
        uint16[3] memory _prizeMultipliers = lotteryList[_lottoID].prizeMultipliers;
        uint16 _rewardMultiplier;
        uint256 _totalReward;
        for (uint8 i = 0; i < ticketList[_lottoID][msg.sender].length; i++) {
            _rewardMultiplier = getRewardMultiplier(
                    getNumberMatching(ticketList[_lottoID][msg.sender][i], lotteryList[_lottoID].winningNumber),
                    _prizeMultipliers
                );
            if (_rewardMultiplier == 0) {
                continue;
            }
            _totalReward += lotteryList[_lottoID].costPerTicket * _rewardMultiplier * 1 ether;
        }
        ULP.sendPrize(msg.sender, _totalReward);
        rewardsClaimed[_lottoID][msg.sender] = true;
        emit ticketsClaimed(currentLottoID, msg.sender);
    }
```

LOCATION	RockPaperScissors.sol -> 118-163

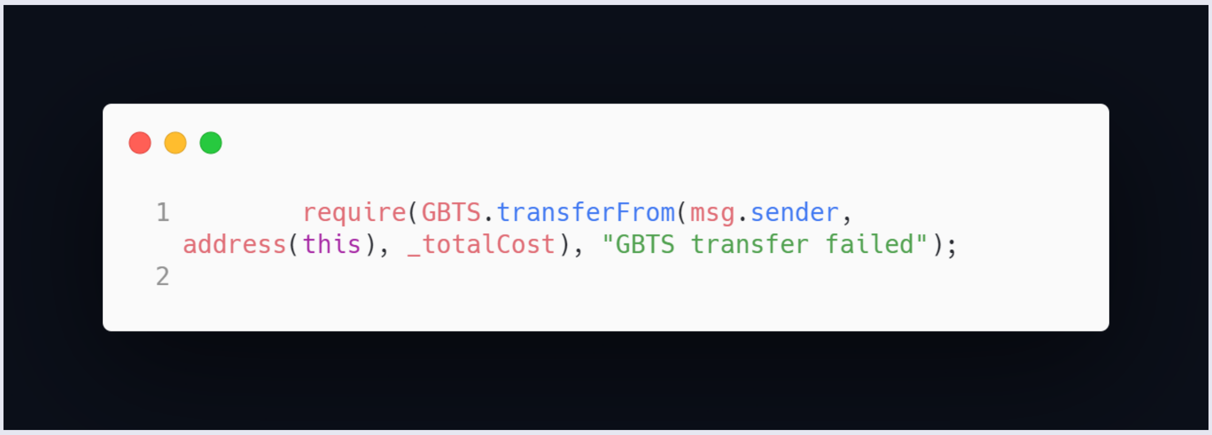


```
function play() external nonReentrant unLocked {
    // ...
    if (gameNumber == betInfo.number) {
        // ...
        ULP.sendPrize(msg.sender, amountToSend);
        // ...
    } else if (
        (gameNumber == 0 && betInfo.number == 1) ||
        (gameNumber == 1 && betInfo.number == 2) ||
        (gameNumber == 2 && betInfo.number == 0)
    ) {
        // ...
        ULP.sendPrize(msg.sender, amountToSend);
        // ...
    } else {
        // ...
    }
    // ...
}
```

DESCRIPTION	If GBTS balance of *UnifiedLiquidityPool* is not enough for prize, prize cannot be claimed.
RECOMMENDATION	Ensure that the contract balance is always greater than the total prizes to be paid out.
MITIGATED/COMMENT	Project team comment: “No real fix to this issue, if a casino goes bankrupt it cannot pay out funds."

Lottery Purchases Paid To Lottery Contract

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0006
LOCATION	Lottery.sol -> 83

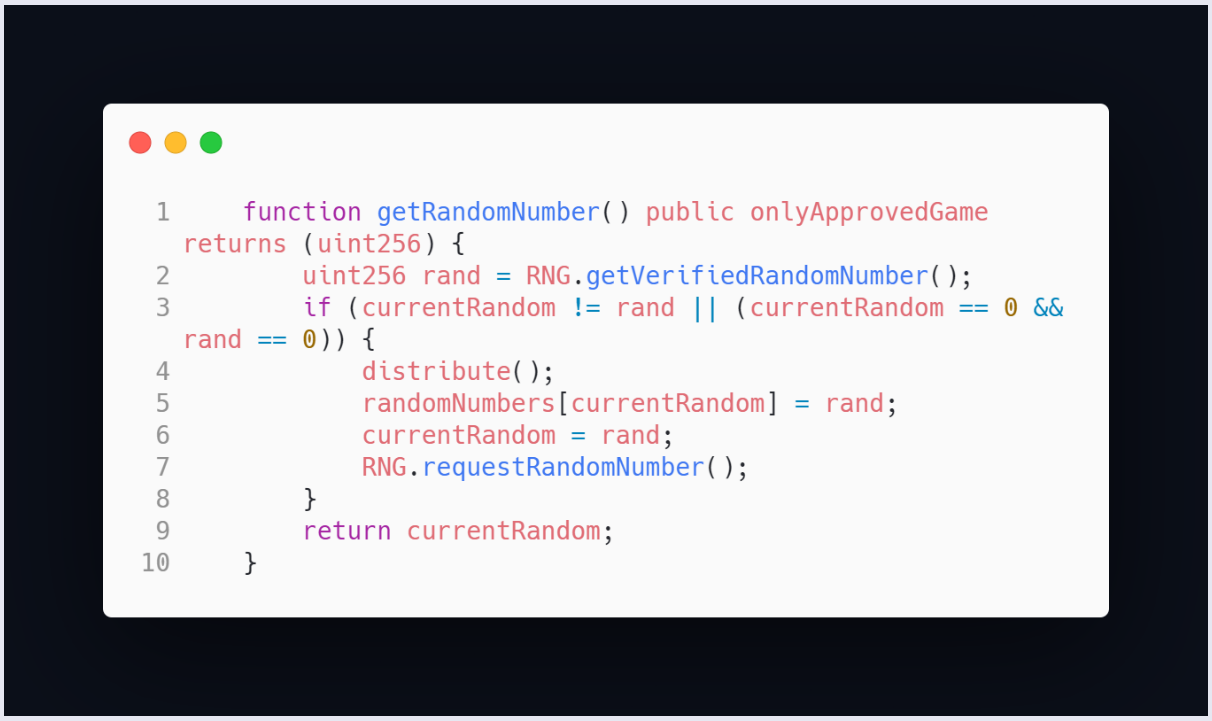


```
        require(GBTS.transferFrom(msg.sender, address(this), _totalCost), "GBTS transfer failed");

```

DESCRIPTION	The lottery contract receives payment for tickets purchased into itself, but pays prizes out from the *UnifiedLiquidityPool*. The GBTS will be locked into the lottery contract forever.
RECOMMENDATION	Direct transfers of GBTS used to purchase lottery tickets to the *UnifiedLiquidityPool*.
MITIGATED/COMMENT	Lottery payments now directed towards *UnifiedLiquidityPool*.

Incorrect Randomness Function Call

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0007
LOCATION	UnifiedLiquidityPool.sol -> 423-432

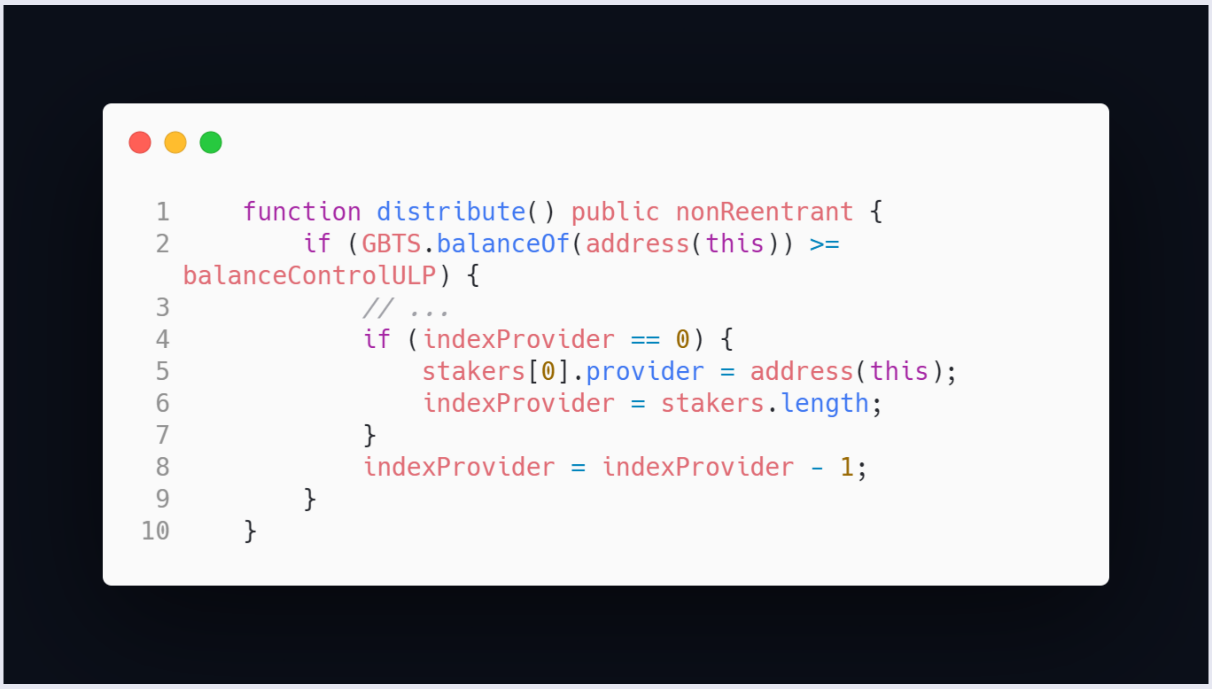


```
    function getRandomNumber() public onlyApprovedGame
returns (uint256) {
        uint256 rand = RNG.getVerifiedRandomNumber();
        if (currentRandom != rand || (currentRandom == 0 &&
rand == 0)) {
            distribute();
            randomNumbers[currentRandom] = rand;
            currentRandom = rand;
            RNG.requestRandomNumber();
        }
        return currentRandom;
    }
```

DESCRIPTION	The function *getVerifiedRandomNumber* requires an argument *bytes32 activeID*. This argument is used to acquire sequentially requested random numbers.
RECOMMENDATION	Re-add *activeRequest* to the *UnifiedLiquidityPool* or change the logic of *getVerifiedRandomNumber* to not require *activeRequest*.
MITIGATED/COMMENT	The active request was added to the call to *RNG*.

Profit Distribution Is Done One At A Time

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0008
LOCATION	UnifiedLiquidityPool.sol -> 307-345

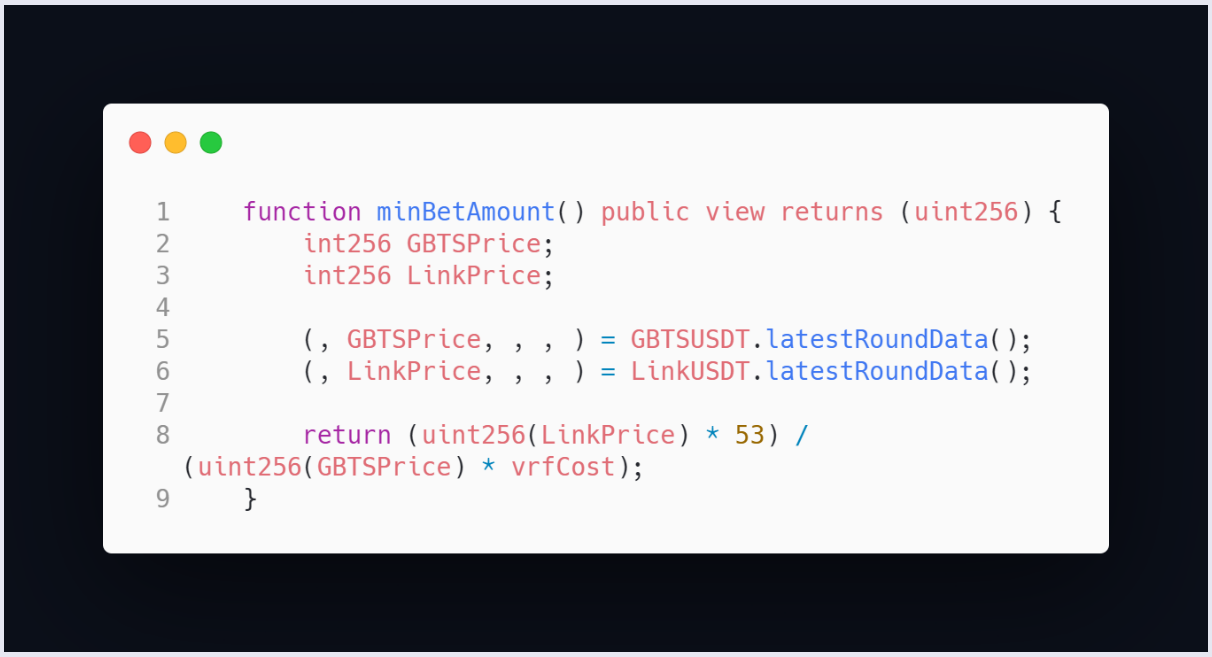


```
function distribute() public nonReentrant {
    if (GBTS.balanceOf(address(this)) >=
balanceControlULP) {
        // ...
        if (indexProvider == 0) {
            stakers[0].provider = address(this);
            indexProvider = stakers.length;
        }
        indexProvider = indexProvider - 1;
    }
}
```

DESCRIPTION	The UnifiedLiquidityPool distributes profits to stakers one at a time. When the number of stakers is very high, this can become very expensive in gas and time consuming. Users might also unstake before receiving their expected rewards.
RECOMMENDATION	Use a reward debt based mechanism to allow users to retrieve their rewards whenever they are available.
MITIGATED/COMMENT	Project team comment: "This is the Gembites business logic. That is for a long period of benefit."

DiceRoll Minimum Bet Value

SEVERITY	Informational
RESOLVED	**NO**
FINDING ID	#0009
LOCATION	DiceRoll.sol -> 162-170 RockPaperScissors.sol -> 168-176

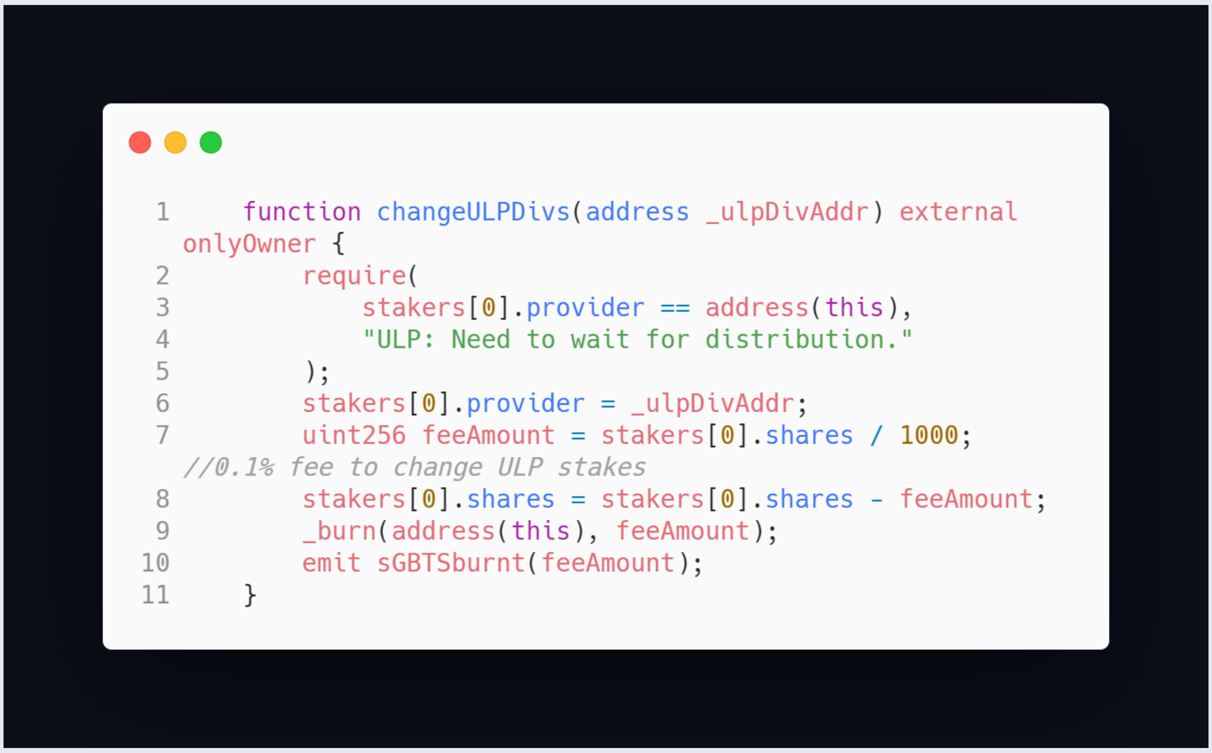


```
    function minBetAmount() public view returns (uint256) {
        int256 GBTSPrice;
        int256 LinkPrice;

        (, GBTSPrice, , , ) = GBTSUSDT.latestRoundData();
        (, LinkPrice, , , ) = LinkUSDT.latestRoundData();

        return (uint256(LinkPrice) * 53) /
  (uint256(GBTSPrice) * vrfCost);
    }
```

DESCRIPTION	The minimum bet of the dice roll contract appears to be 0.0053 LINK in GBTS.
RECOMMENDATION	Confirm that this is the expected minimum bet.
MITIGATED/COMMENT	Project team has confirmed this is the correct minimum bet.

Address Of Zero Staker Can Be Modified

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0010
LOCATION	UnifiedLiquidityPool.sol -> 353-363

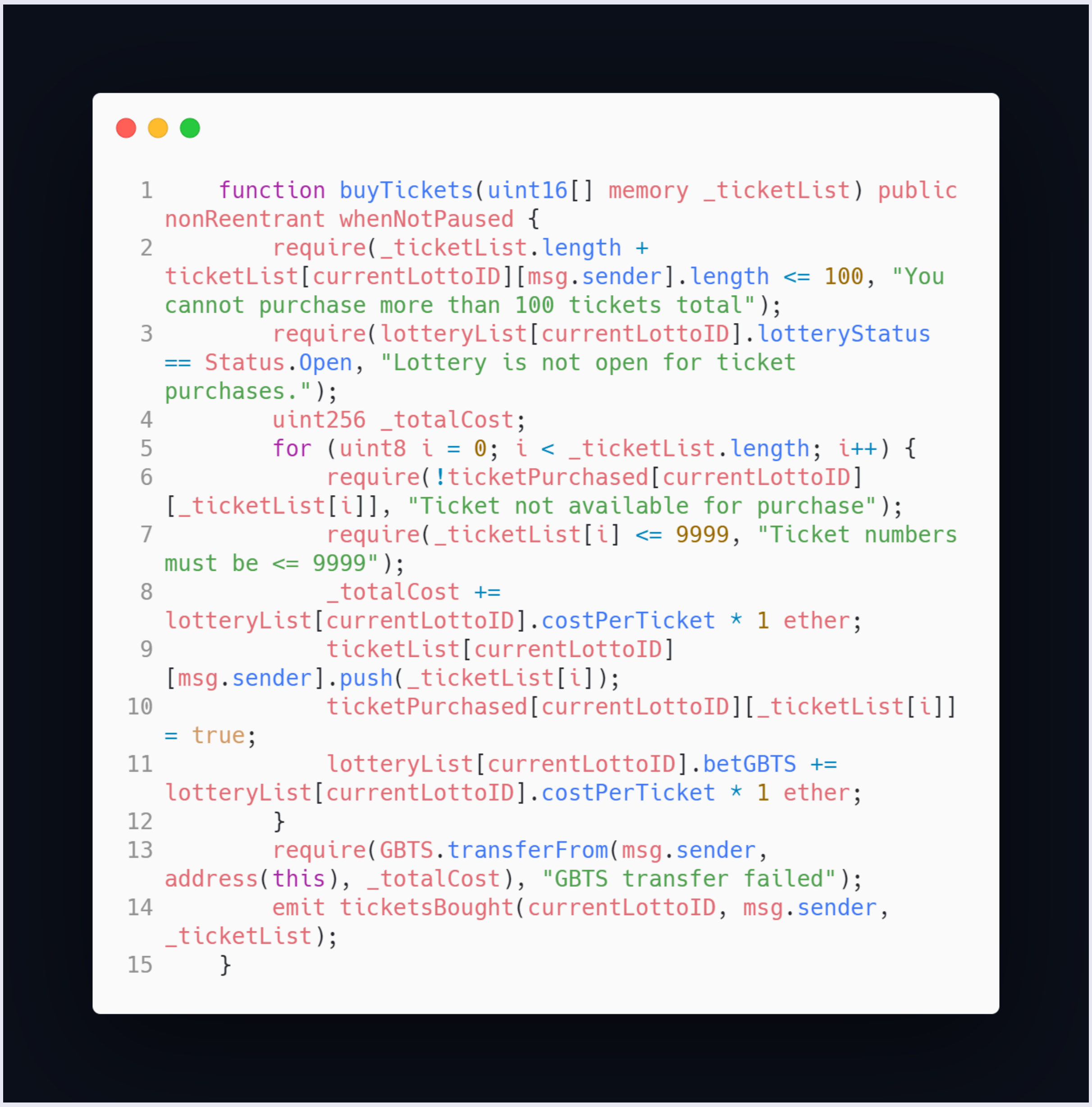


```
    function changeULPDivs(address _ulpDivAddr) external onlyOwner {
        require(
            stakers[0].provider == address(this),
            "ULP: Need to wait for distribution."
        );
        stakers[0].provider = _ulpDivAddr;
        uint256 feeAmount = stakers[0].shares / 1000; //0.1% fee to change ULP stakes
        stakers[0].shares = stakers[0].shares - feeAmount;
        _burn(address(this), feeAmount);
        emit sGBTSburnt(feeAmount);
    }
```

DESCRIPTION	The zero indexed staker can have its address modified. It will generally be set to the contract address and will be reset after a distribution occurs. However, this can be used to add more shares to the zero indexed staker via *addToDividendPool*. This is an atypical way to enable that functionality. Furthermore, if distribution does not occur, it cannot be reset until the next distribution.
RECOMMENDATION	Do not allow the reassignment of the provider address. Instead directly expose the desired functionality (eg. Add to contract stake).
MITIGATED/COMMENT	The project team has disabled the ability to add more shares to the zero indexed staker.

Ticket Limit Can Be Bypassed

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0011
LOCATION	Lottery.sol -> 71-85



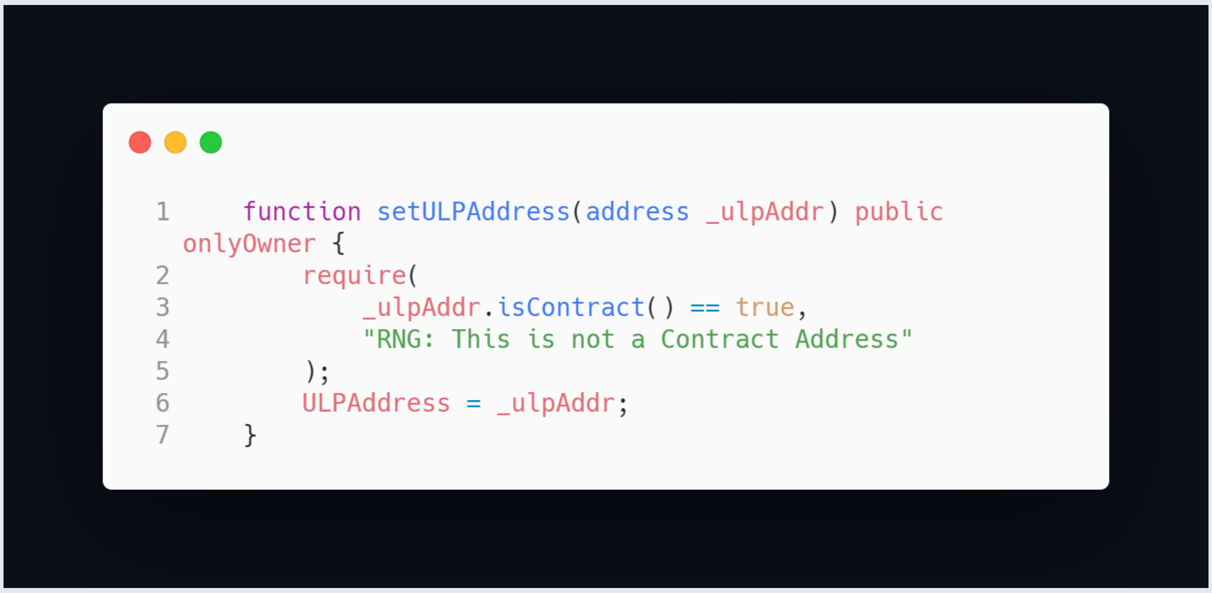
```
    function buyTickets(uint16[] memory _ticketList) public nonReentrant whenNotPaused {
        require(_ticketList.length + ticketList[currentLottoID][msg.sender].length <= 100, "You cannot purchase more than 100 tickets total");
        require(lotteryList[currentLottoID].lotteryStatus == Status.Open, "Lottery is not open for ticket purchases.");
        uint256 _totalCost;
        for (uint8 i = 0; i < _ticketList.length; i++) {
            require(!ticketPurchased[currentLottoID][_ticketList[i]], "Ticket not available for purchase");
            require(_ticketList[i] <= 9999, "Ticket numbers must be <= 9999");
            _totalCost += lotteryList[currentLottoID].costPerTicket * 1 ether;
            ticketList[currentLottoID][msg.sender].push(_ticketList[i]);
            ticketPurchased[currentLottoID][_ticketList[i]] = true;
            lotteryList[currentLottoID].betGBTS += lotteryList[currentLottoID].costPerTicket * 1 ether;
        }
        require(GBTS.transferFrom(msg.sender, address(this), _totalCost), "GBTS transfer failed");
        emit ticketsBought(currentLottoID, msg.sender, _ticketList);
    }
```

DESCRIPTION	Lottery contract checks that a given address does not purchase more than 100 tickets, however, users may use multiple wallets or proxy contracts to bypass this.
RECOMMENDATION	Design the contract in such a way that users exceeding the limit will not be detrimental instead of putting a direct limit.

MITIGATED/COMMENT	Project team comment: “the token limit is a measure to prevent iteration from going over the maximum gas amount. It has nothing to do with preventing a single person from buying > 100 tickets”

No Events Emitted For Changes To Protocol Values

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0012
LOCATION	RandomNumberConsumer.sol -> 103-109

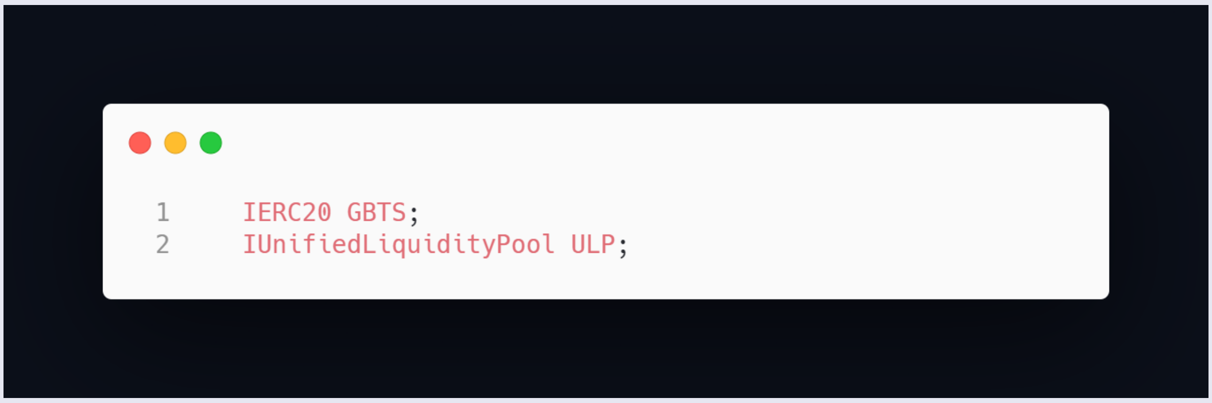


```
    function setULPAddress(address _ulpAddr) public
onlyOwner {
        require(
            _ulpAddr.isContract() == true,
            "RNG: This is not a Contract Address"
        );
        ULPAddress = _ulpAddr;
    }
```

DESCRIPTION	Functions that change important variables should include emit logs such that users can more easily monitor the change.
RECOMMENDATION	Add emit logs to these functions. Ensure that these values are secured via a timelock.
MITIGATED/COMMENT	An event was added.

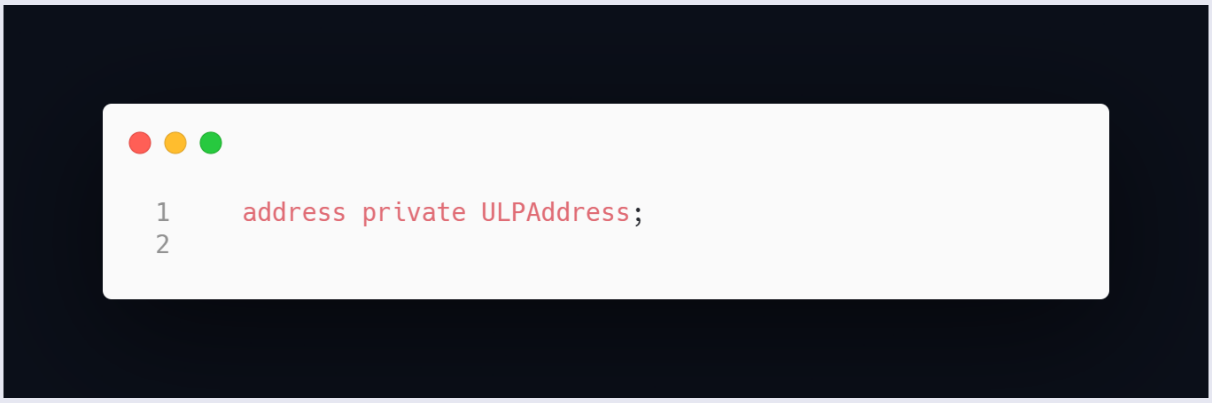
Protocol Values Should Be Public

SEVERITY	Informational
RESOLVED	**PARTIAL**
FINDING ID	#0013
LOCATION	Lottery.sol -> 44-45



```
IERC20 GBTS;
IUnifiedLiquidityPool ULP;
```

LOCATION	RandomNumberConsumer.sol -> 19

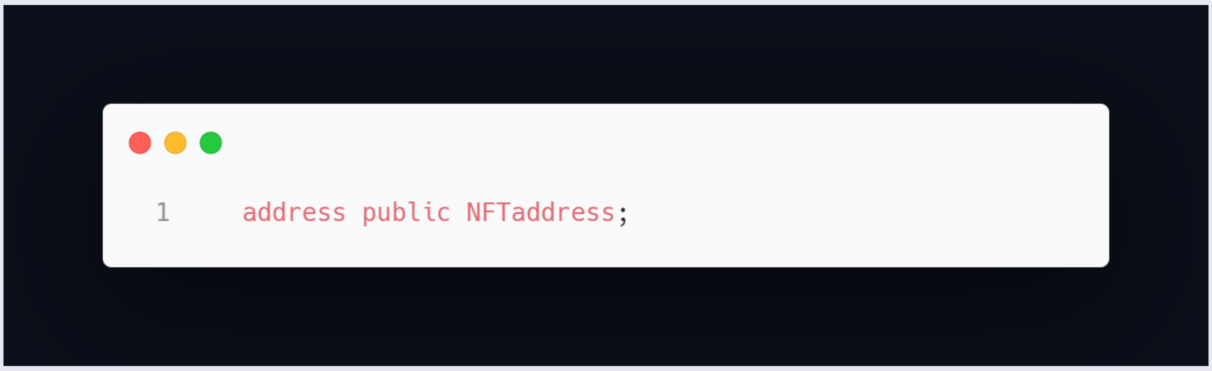


```
address private ULPAddress;

```

DESCRIPTION	Variables critical to the operation of the protocol should be public or have an associated view function.
RECOMMENDATION	Change the value to be public.
MITIGATED/COMMENT	The value of *ULPAddress* in *RandomNumberConsumer* was made public.

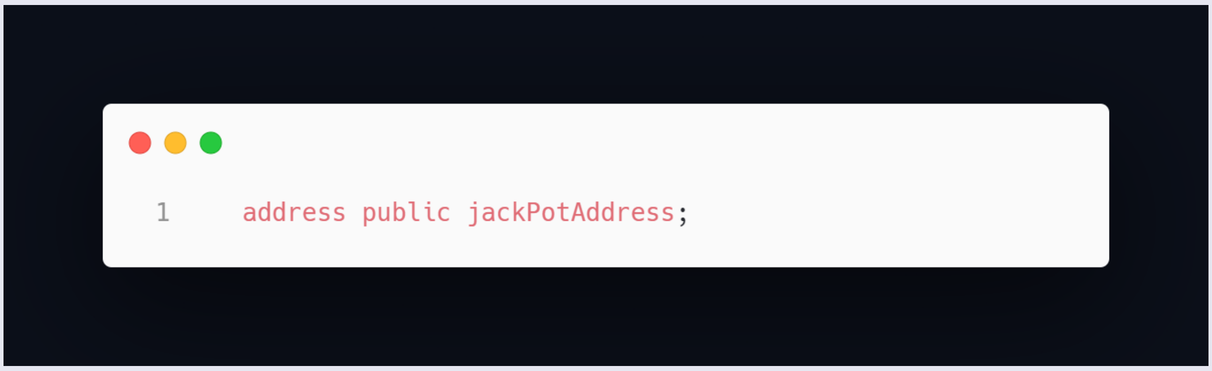
Contract Variables Set But Never Used

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0014
LOCATION	UnifiedLiquidityPool.sol -> 76



```
address public NFTaddress;
```

LOCATION	UnifiedLiquidityPool.sol -> 82

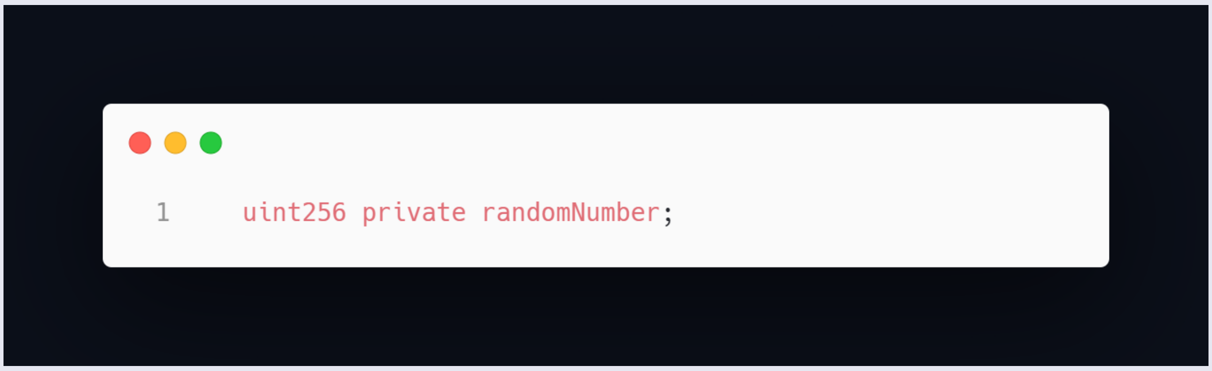


```
address public jackPotAddress;
```

DESCRIPTION	Contract values are set but never used.
RECOMMENDATION	Remove these variables and associated setters.
MITIGATED/COMMENT	Variables and associated setters were removed.

Contract Variable Used But Never Set

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0015
LOCATION	RandomNumberConsumer.sol -> 18



```
uint256 private randomNumber;
```

DESCRIPTION	Contract value is used but never set. This variable is only used to emit events so will have no impact on the operation of the contract.
RECOMMENDATION	Remove this variable and change the event to emit useful values.
MITIGATED/COMMENT	Variable was removed, the event now emits the received random numbers.

Unused Events

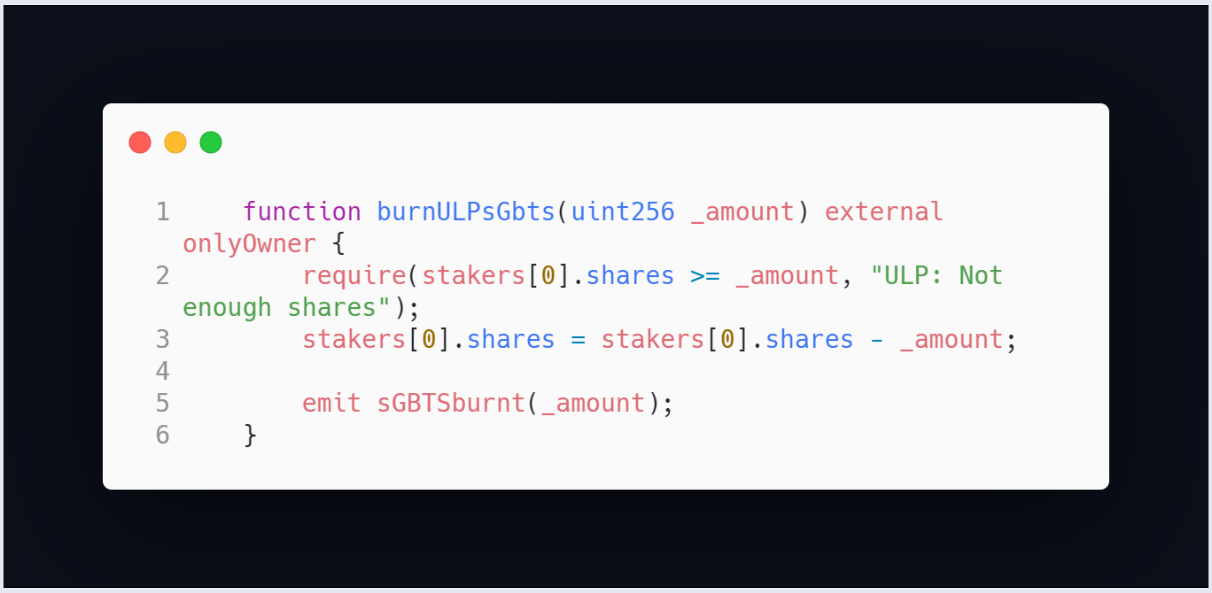
SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0016
LOCATION	UnifiedLiquidityPool.sol -> 28

```
event RNGChanged(address rngAddr);
```

DESCRIPTION	This event is never emitted.
RECOMMENDATION	Remove the event.
MITIGATED/COMMENT	The event was removed.

Shares Only Deducted, Not Burned

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0017
LOCATION	UnifiedLiquidityPool.sol -> 482-487

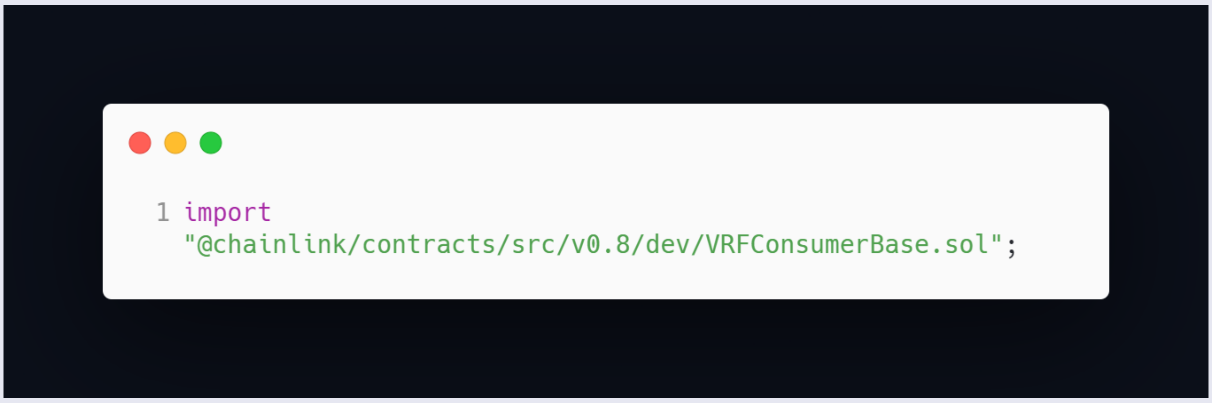


```
    function burnULPsGbts(uint256 _amount) external onlyOwner {
        require(stakers[0].shares >= _amount, "ULP: Not enough shares");
        stakers[0].shares = stakers[0].shares - _amount;

        emit sGBTSburnt(_amount);
    }
```

DESCRIPTION	Comments imply this function is intended to burn shares. However, it only deducts the shares from the staked balance of the contract itself.
RECOMMENDATION	Confirm that this behaviour is intended.
MITIGATED/COMMENT	Contract updated to burn the tokens.

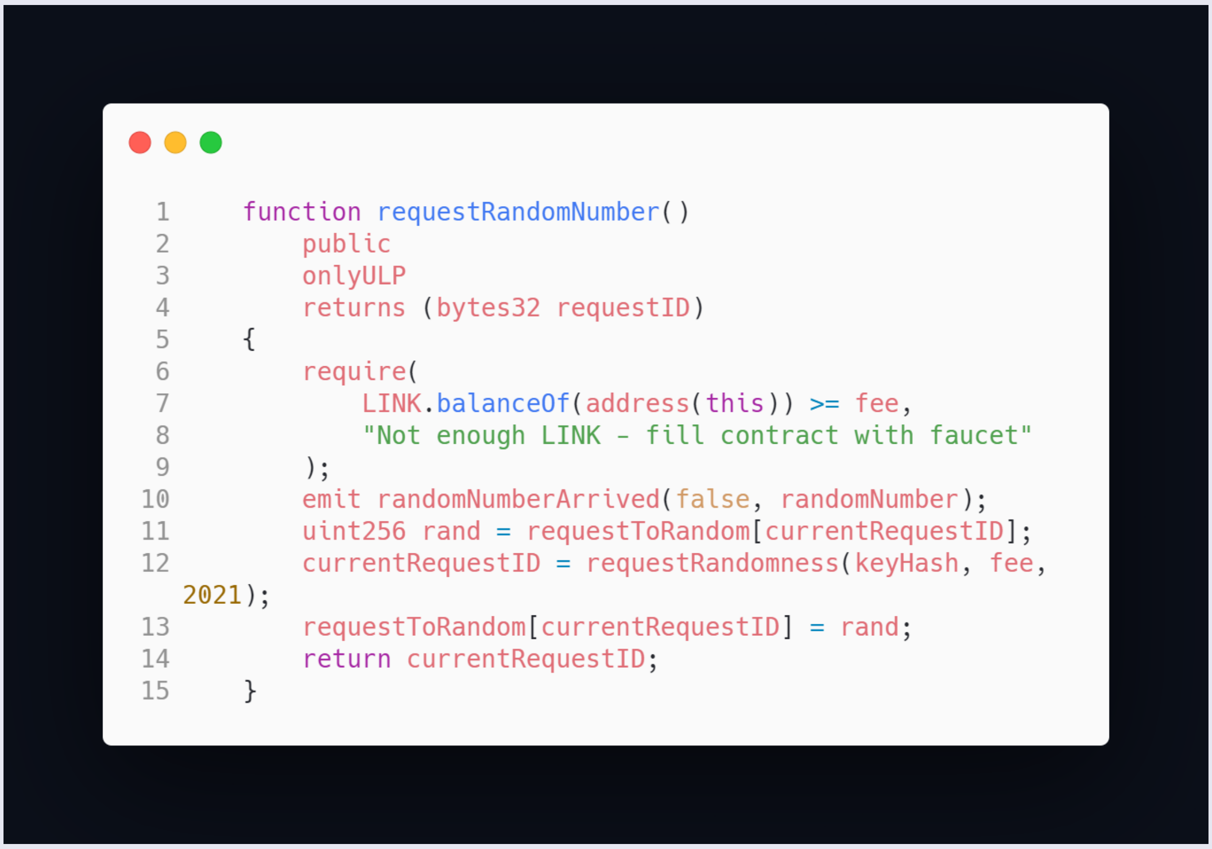
Chainlink VRF Call Does Not Match Signature

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0018
LOCATION	RandomNumberConsumer.sol -> 4



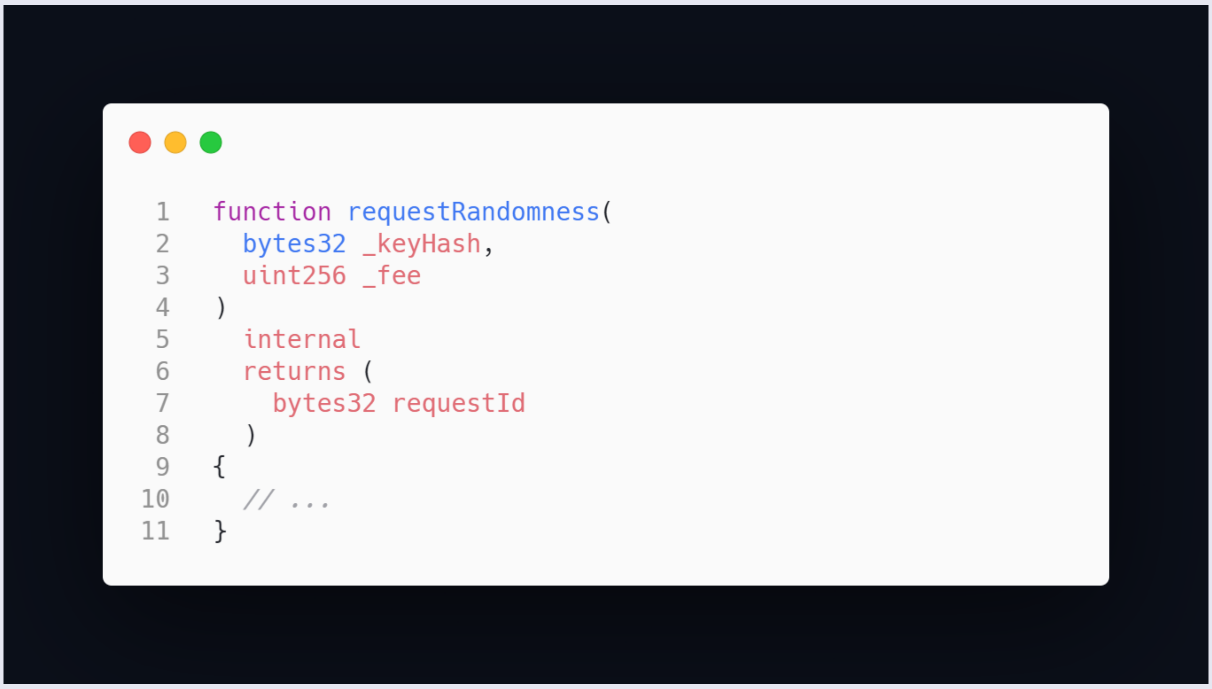
```
import "@chainlink/contracts/src/v0.8/dev/VRFConsumerBase.sol";
```

LOCATION	RandomNumberConsumer.sol -> 59-73



```
    function requestRandomNumber()
        public
        onlyULP
        returns (bytes32 requestID)
    {
        require(
            LINK.balanceOf(address(this)) >= fee,
            "Not enough LINK - fill contract with faucet"
        );
        emit randomNumberArrived(false, randomNumber);
        uint256 rand = requestToRandom[currentRequestID];
        currentRequestID = requestRandomness(keyHash, fee, 2021);
        requestToRandom[currentRequestID] = rand;
        return currentRequestID;
    }
```

LOCATION	@chainlink/contracts/src/v0.8/dev/VRFConsumerBase.sol -> 158-180

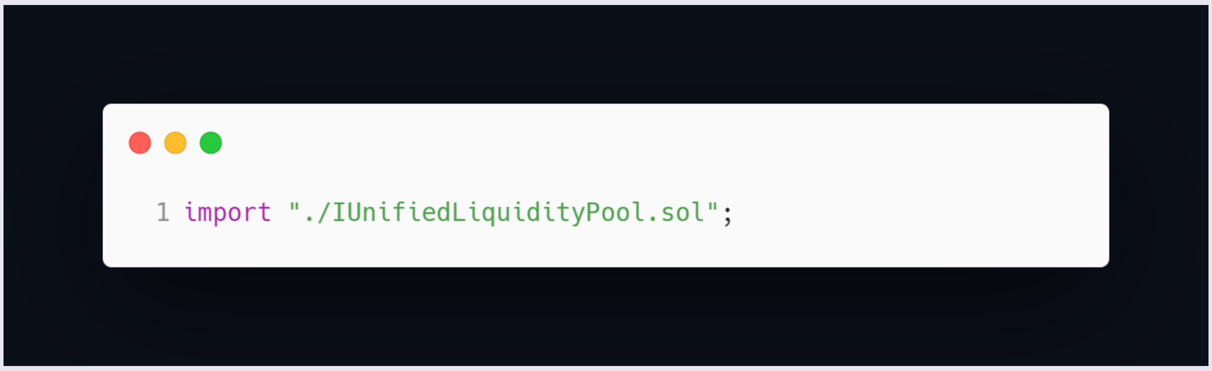


```
function requestRandomness(
  bytes32 _keyHash,
  uint256 _fee
)
  internal
  returns (
    bytes32 requestId
  )
{
  // ...
}
```

DESCRIPTION	Call to *requestRandomness* does not match signature in base contract. A constant is passed as a user seed, which is unnecessary, as stated by ChainLink this argument is deprecated.
RECOMMENDATION	Verify that function parameters match the expected inputs. Remove the *userProvidedSeed* from *getRandomNumber()*.
MITIGATED/COMMENT	The function was modified so that the proper parameters are used.

Invalid Import Path

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0019
LOCATION	Lottery.sol -> 8

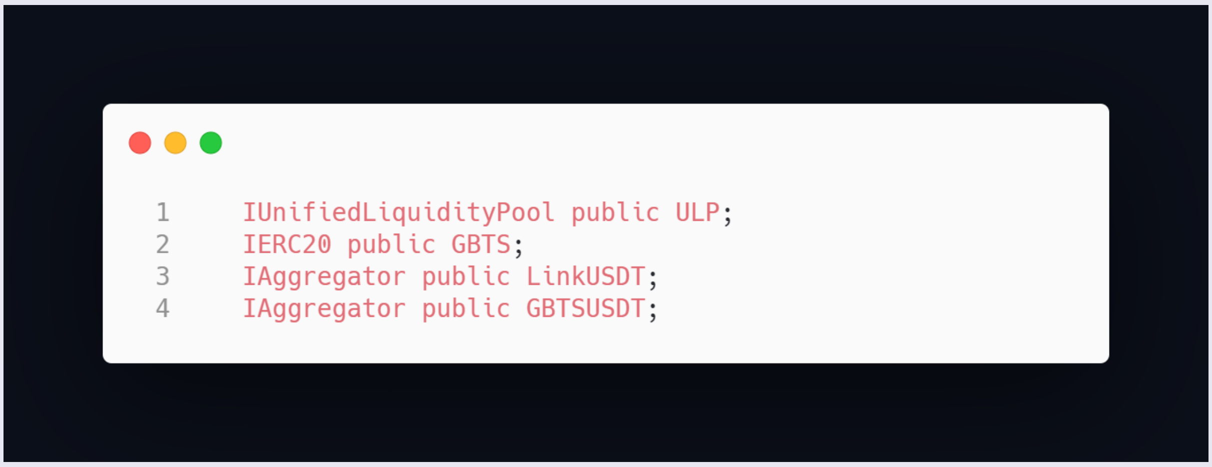


```
import "./IUnifiedLiquidityPool.sol";
```

DESCRIPTION	Import path does not match location of contract file.
RECOMMENDATION	Correct path to "./interfaces/IUnifiedLiquidityPool.sol".
MITIGATED/COMMENT	Import path was fixed.

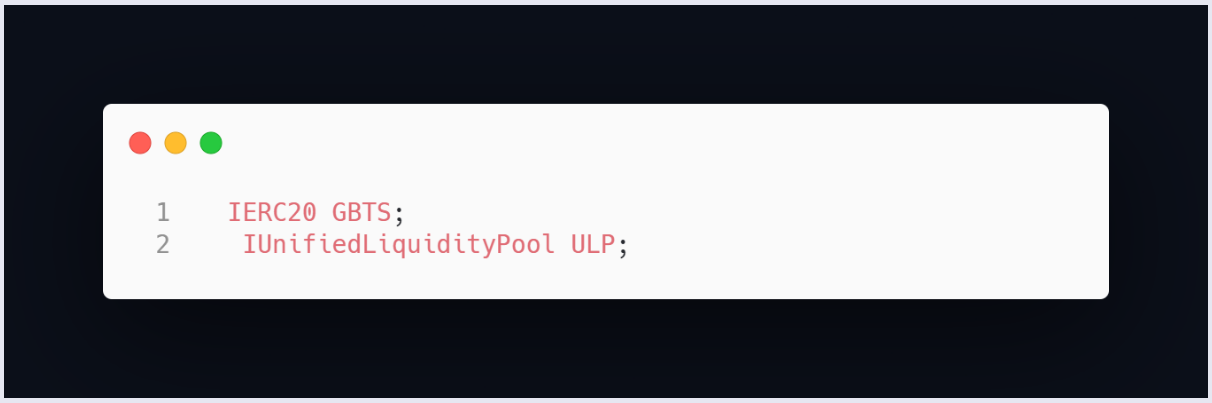
Contract Value Can Be Constant or Immutable

SEVERITY	Informational
RESOLVED	**PARTIAL**
FINDING ID	#0020
LOCATION	DiceRoll.sol -> 14-17 RockPaperScissors.sol -> 14-17



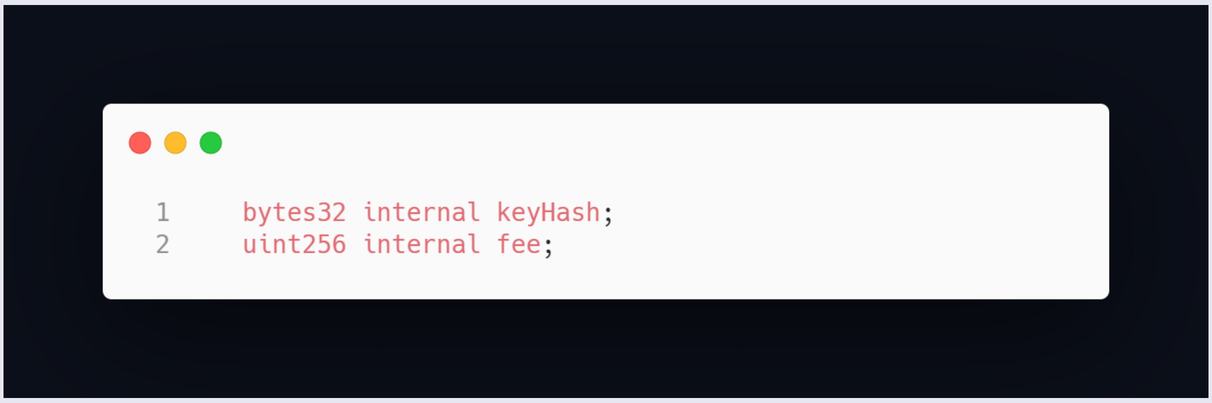
```
IUnifiedLiquidityPool public ULP;
IERC20 public GBTS;
IAggregator public LinkUSDT;
IAggregator public GBTSUSDT;
```

LOCATION	Lottery.sol -> 44-45



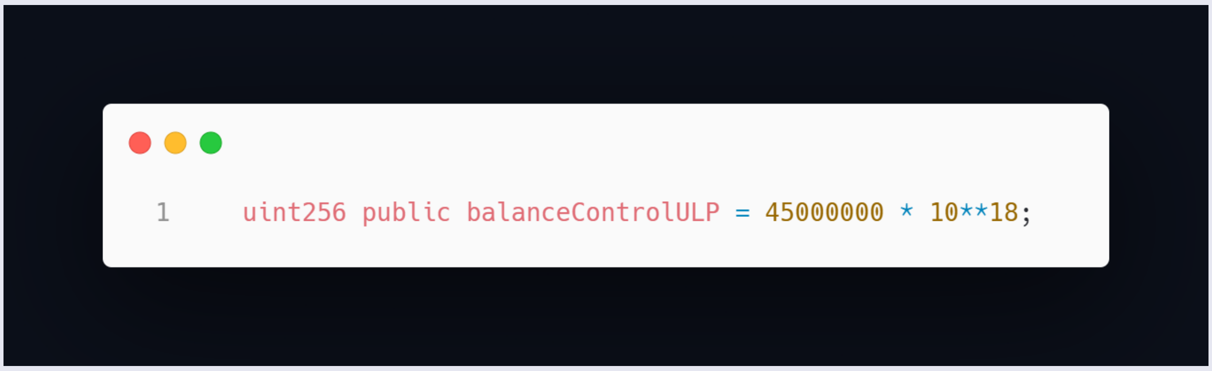
```
IERC20 GBTS;
 IUnifiedLiquidityPool ULP;
```

LOCATION	RandomNumberConsumer.sol -> 15-16



```
bytes32 internal keyHash;
uint256 internal fee;
```

LOCATION	UnifiedLiquidityPool.sol -> 97

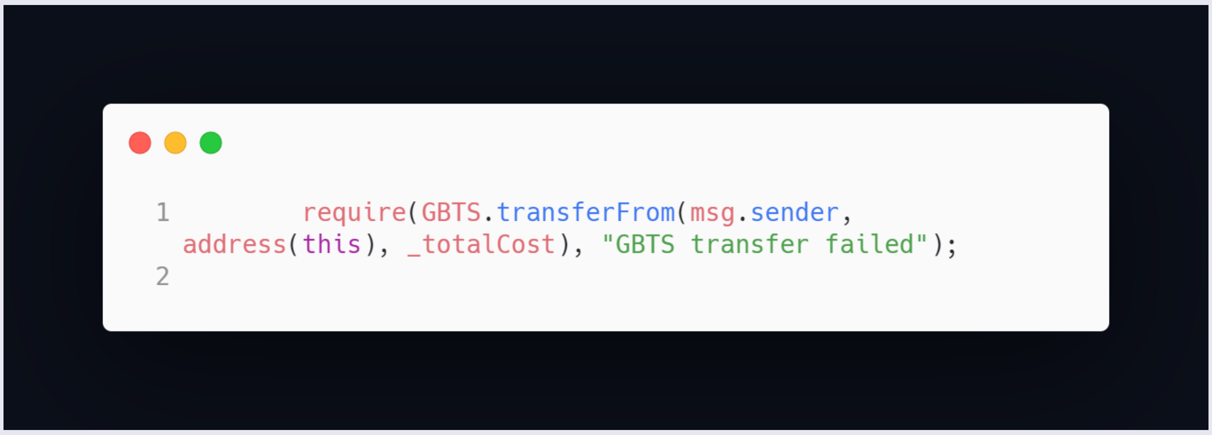


```
uint256 public balanceControlULP = 45000000 * 10**18;
```

DESCRIPTION	The noted contract values never change.
RECOMMENDATION	Change the variables to be constant.
MITIGATED/COMMENT	Value of *UnifiedLiquidityPool.balanceControlULP* was made constant.

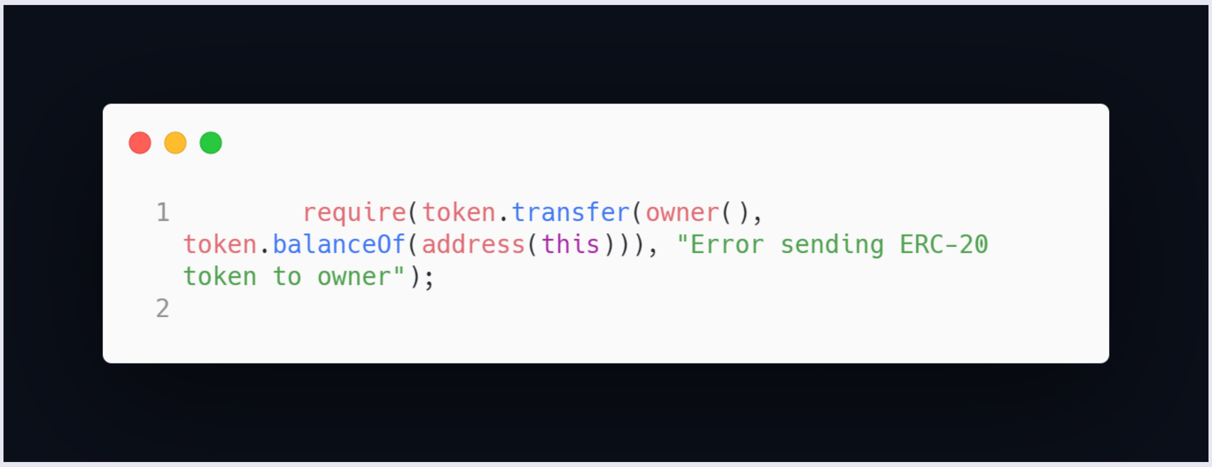
Use Safe Transfer

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0021
LOCATION	Lottery.sol -> 83



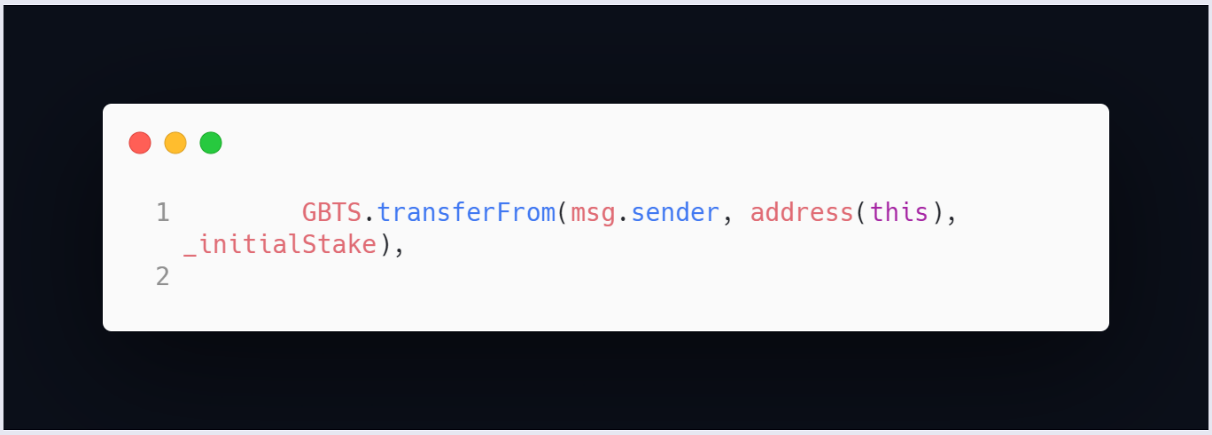
```
        require(GBTS.transferFrom(msg.sender, address(this), _totalCost), "GBTS transfer failed");
```

LOCATION	Lottery.sol -> 191



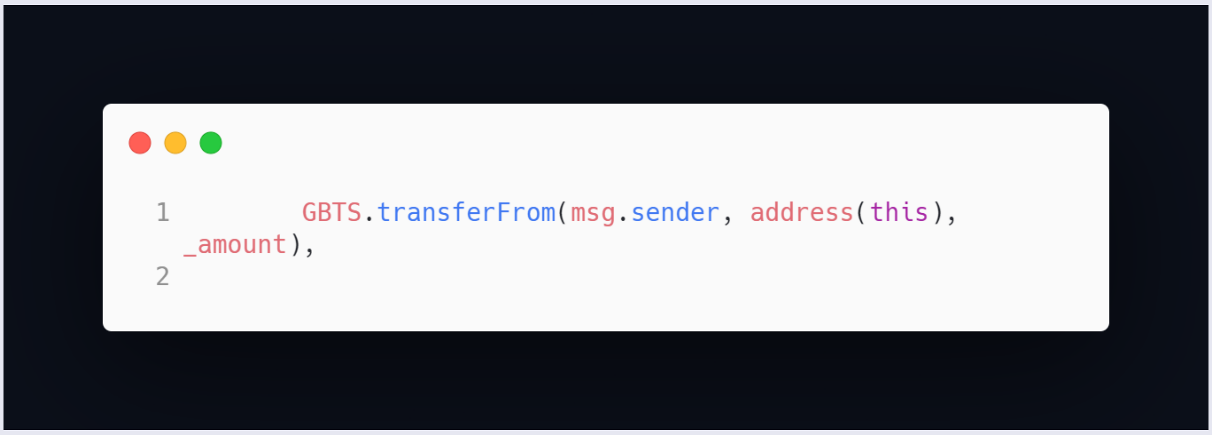
```
        require(token.transfer(owner(), token.balanceOf(address(this))), "Error sending ERC-20 token to owner");
```

LOCATION	UnifiedLiquidityPool.sol -> 173



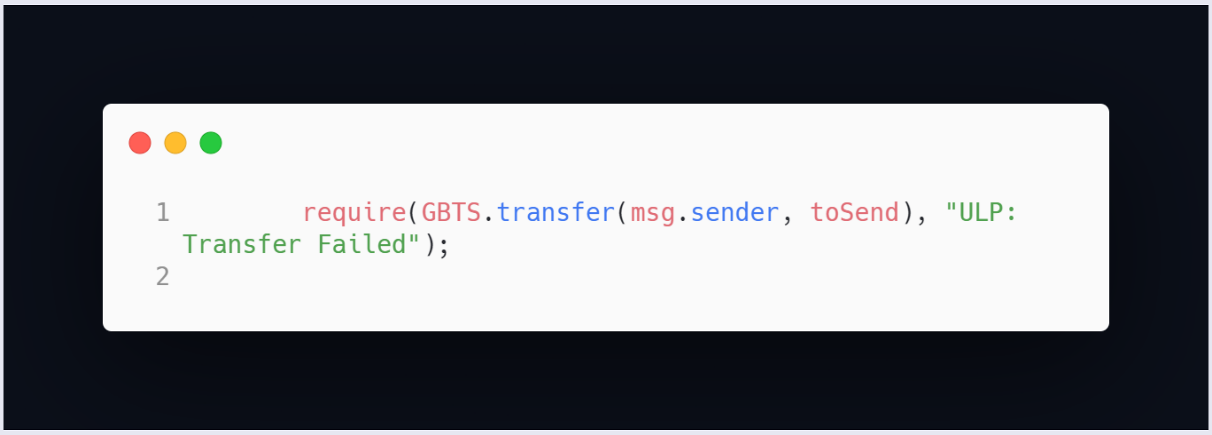
```
        GBTS.transferFrom(msg.sender, address(this), _initialStake),
```

LOCATION	UnifiedLiquidityPool.sol -> 201



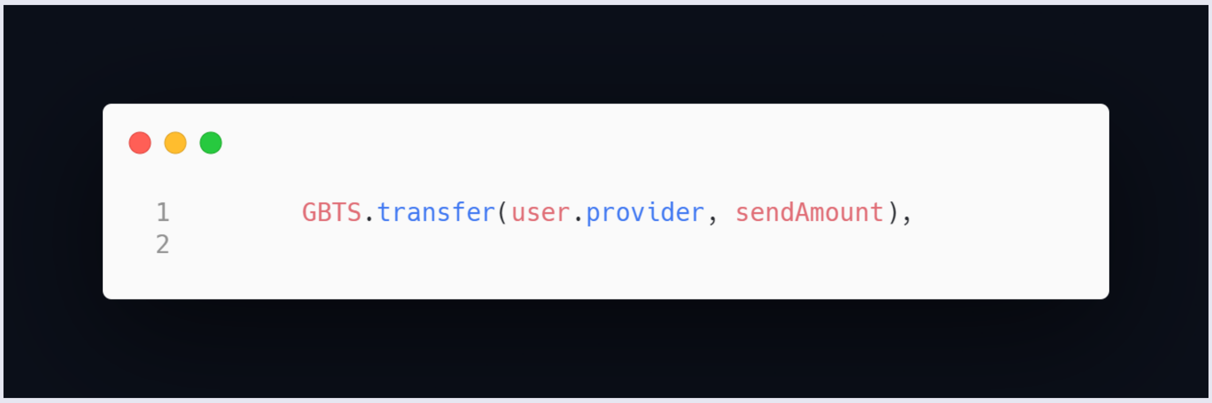
```
        GBTS.transferFrom(msg.sender, address(this), _amount),
```

LOCATION	UnifiedLiquidityPool.sol -> 231



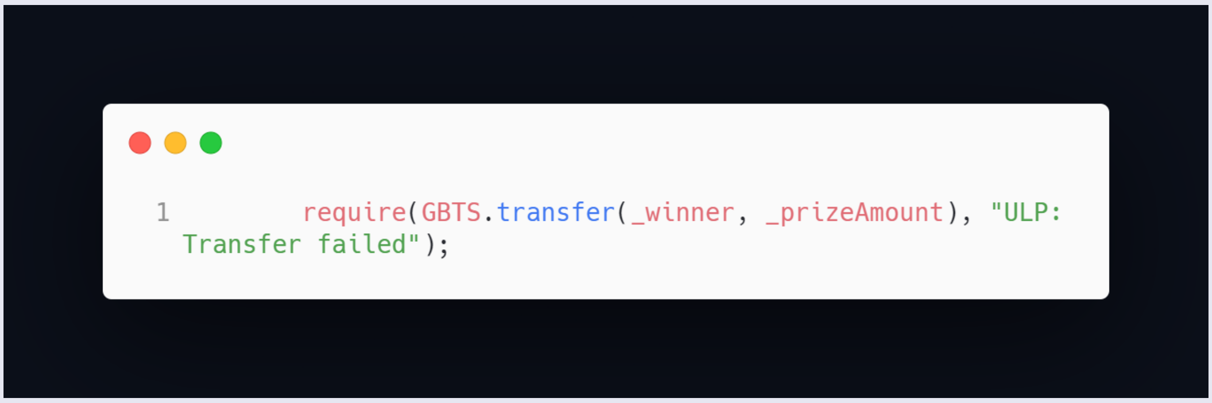
```
        require(GBTS.transfer(msg.sender, toSend), "ULP: Transfer Failed");
```

LOCATION	UnifiedLiquidityPool.sol -> 330



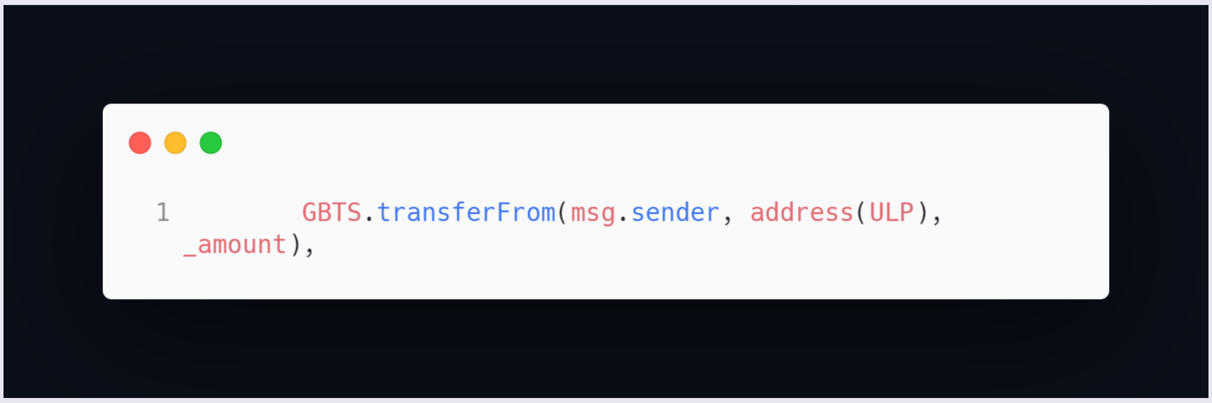
```
        GBTS.transfer(user.provider, sendAmount),
```

LOCATION	UnifiedLiquidityPool.sol -> 415



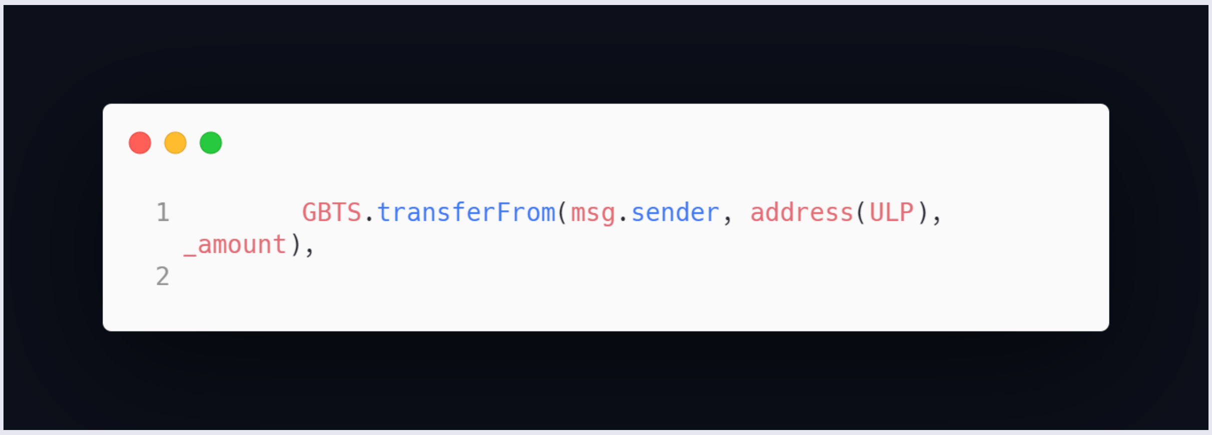
```
        require(GBTS.transfer(_winner, _prizeAmount), "ULP: Transfer failed");
```

LOCATION	RockPaperScissors.sol -> 103



```
        GBTS.transferFrom(msg.sender, address(ULP), _amount),
```

LOCATION	DiceRoll.sol -> 103

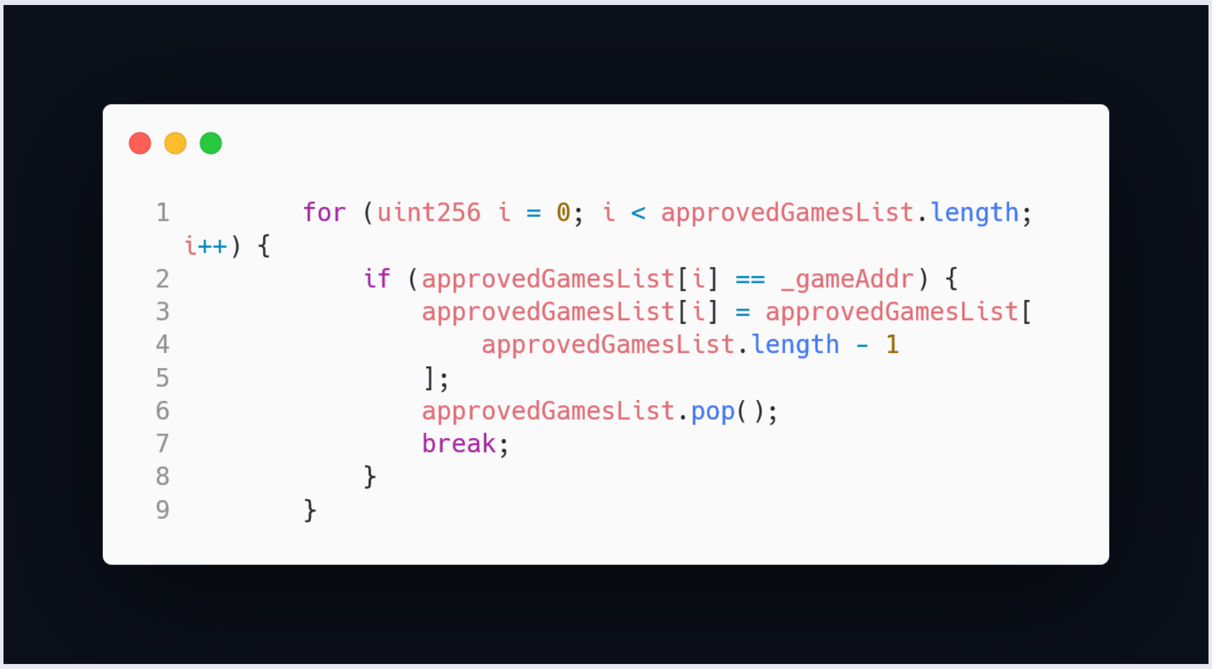


```
        GBTS.transferFrom(msg.sender, address(ULP), _amount),

```

DESCRIPTION	Direct transfer functions are called.
RECOMMENDATION	Use openzeppelin's safe transfer functions. These safe transfer function are used to catch when a transfer fails as well as unusual token behaviour.
MITIGATED/COMMENT	Project team has implemented the recommended fix.

Unbound Loop

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0023
LOCATION	UnifiedLiquidityPool.sol -> 369



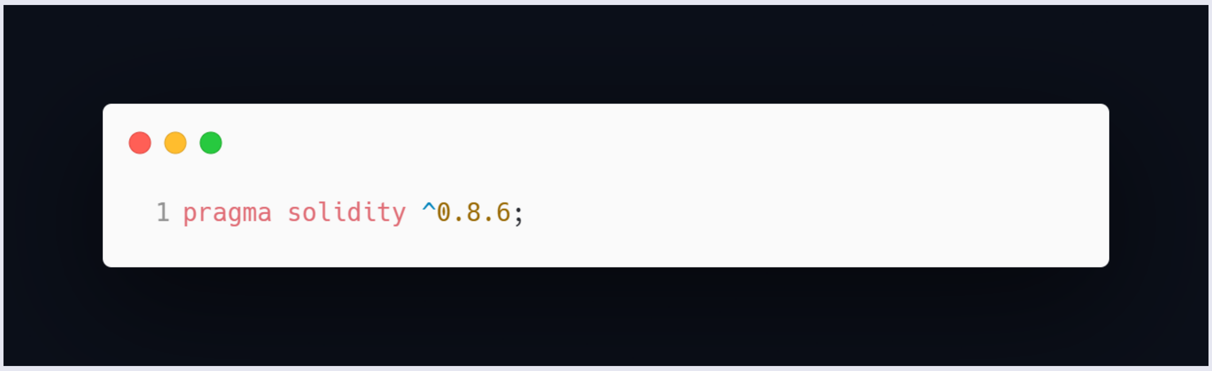
```
        for (uint256 i = 0; i < approvedGamesList.length; i++) {
            if (approvedGamesList[i] == _gameAddr) {
                approvedGamesList[i] = approvedGamesList[
                    approvedGamesList.length - 1
                ];
                approvedGamesList.pop();
                break;
            }
        }
```

DESCRIPTION	There is no limit to the number of games which can be added. If enough games are added, all subsequent calls to *changeGameApproval* will revert due to the transaction gas limit.
RECOMMENDATION	Add a maximum number of approved games.
MITIGATED/COMMENT	Implementation of *approvedGamesList* was changed to not use a loop.

Static Analysis

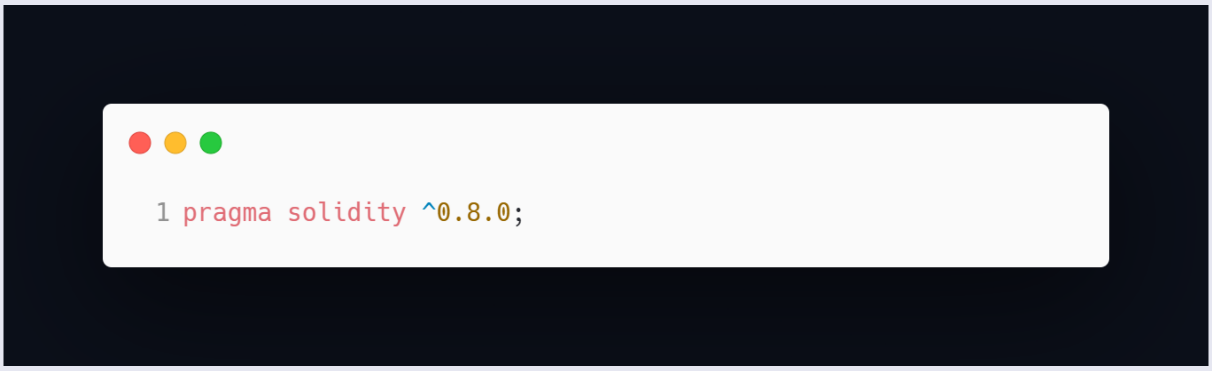
Different Versions Of Solidity

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0022
LOCATION	DiceRoll.sol -> 2 IAggregator.sol -> 2 IRandomNumberConsumer.sol -> 2 IUnifiedLiquidityPool.sol -> 2 RandomNumberConsumer.sol -> 2 UnifiedLiquidityPool.sol -> 2



```
pragma solidity ^0.8.6;
```

LOCATION	Lottery.sol -> 1



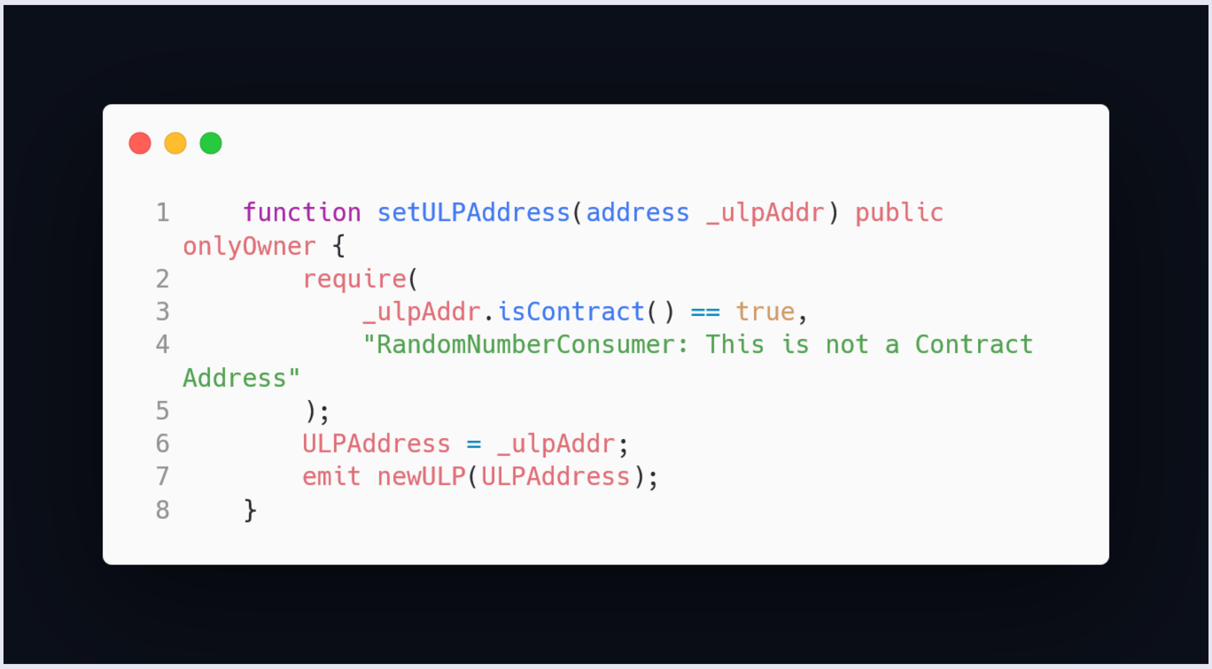
```
pragma solidity ^0.8.0;
```

DESCRIPTION	Different versions of solidity are used.
RECOMMENDATION	Stick with one Solidity version.
MITIGATED/COMMENT	All contracts changed to use Solidity 0.8.6.

On-Chain Analysis

RandomNumberConsumer can be disconnected from ULP

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0024
LOCATION	RandomNumberConsumer.sol -> 110 RandomNumberConsumer: 0x2B3701955C6d6B4d134F84DA43f480A97829020F -> 754



```
    function setULPAddress(address _ulpAddr) public onlyOwner {
        require(
            _ulpAddr.isContract() == true,
            "RandomNumberConsumer: This is not a Contract Address"
        );
        ULPAddress = _ulpAddr;
        emit newULP(ULPAddress);
    }
```

DESCRIPTION	The contract owner of RandomNumberConsumer can change the address value *ULP*, breaking the functionality of the *UnifiedLiquidityPool* contract.
RECOMMENDATION	Renounce ownership of the contract or transfer ownership of the contract to a timelock.
MITIGATED/COMMENT	Contract ownership was renounced

Multiple Versions of OpenZeppelin

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0025
LOCATION	RandomNumberConsumer: 0x2B3701955C6d6B4d134F84DA43f480A97829020F UnifiedLiquidityPool: 0xbD658acCb3364b292E2f7620F941d4662Fd25749

DESCRIPTION	Multiple versions of OpenZeppelin were used. In general, it is recommended to use a single consistent version of any given library within a single project. UnifiedLiquidityPool uses OpenZeppelin 4.1.0 while RandomNumberConsumer uses OpenZeppelin 4.2.0.
RECOMMENDATION	No change necessary.
MITIGATED/COMMENT	N/A

Appendix A - Reviewed Documents

Document	**Address**
DiceRoll.sol	N/A
IAggregator.sol	N/A
IRandomNumberConsumer.sol	N/A
IUnifiedLiquidityPool.sol	N/A
Lottery.sol	N/A
RandomNumberConsumer.sol	0x2B3701955C6d6B4d134F84DA43f480A97829020F
RockPaperScissors.sol	N/A
UnifiedLiquidityPool.sol	0xbD658acCb3364b292E2f7620F941d4662Fd25749

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause immediate loss of Tokens / Funds
Medium Risk	A vulnerability that can cause some loss of Tokens / Funds
Low Risk	A vulnerability that can be mitigated
Informational	No vulnerability

Appendix C - Icons

Icon	Explanation
📋	Solved by Project Team
🔍	Under Investigation of Project Team
⚠	Unsolved

Appendix D - Testing Standard

An ordinary audit is conducted using these steps.

1. Gather all information
2. Conduct a first visual inspection of documents and contracts
3. Go through all functions of the contract manually (2 independent auditors)
    a. Discuss findings
4. Use specialized tools to find security flaws
    a. Discuss findings
5. Follow up with project lead of findings
6. If there are flaws, and they are corrected, restart from step 2
7. Write and publish a report

During our audit, a thorough investigation has been conducted employing both automated analysis and manual inspection techniques. Our auditing method lays a particular focus on the following important concepts:

- Ensuring that the code and codebase use best practices, industry standards, and available libraries.
- Testing the contract from different angles ensuring that it works under a multitude of circumstances.
- Analyzing the contracts through databases of common security flaws.

Follow Obelisk Auditing for the Latest Information

ObeliskOrg ObeliskOrg

OBELISK

Part of Tibereum Group