OKLG - Smart Contract Audit Report

AUDIT SUMMARY

Ok.let's.go is creating a platform with tokens and various services that can be purchased by users, allowing them to easily create their own Token Lockers, Token Bridges, Raffles, Staking platforms, and more.

For this audit, we reviewed the select contracts listed below at commite167d0d742d21bcc62d7a5b770a1f0ed1cf31eca on the team's GitHub repository.

AUDIT FINDINGS

> *Please ensure trust in the team and users of the team's products prior to investing as they have substantial control in the ecosystem.*
> *Date: January 14th, 2022.*
> *Updated: February 4th, 2022 to reflect changes from commit393190aee594178f2a9e7f8646cb9b3bb09507f9 to commit 28447bcbaf453737d430569313c2850d707ec79c.*
> *Updated: February 14th, 2022 to reflect changes from commit28447bcbaf453737d430569313c2850d707ec79c to commit c09c970301bb91c8af21a3c177b220375e17480b.*
> *Updated: February 17th, 2022 to reflect changes from commitc09c970301bb91c8af21a3c177b220375e17480b to commit e167d0d742d21bcc62d7a5b770a1f0ed1cf31eca.*

OKLGTokenLocker - Finding #1 - High (Resolved)

Description: *In the withdrawLockedTokens() function, there is no check to confirm that the Locker's end time has passed when withdrawing NFTs, and improperly checks the amount to be withdrawn when withdrawing ERC20 tokens.*

```
if (_locker.isNft) {
  IERC721 _token = IERC721(_locker.token);
  _token.transferFrom(address(this), msg.sender, _amountOrTokenId);
} else {
  uint256 _maxAmount = maxWithdrawableTokens(_idx);
  require(
 _amountOrTokenId > 0 &&
   _maxAmount > 0 &&
   _maxAmount <= _amountOrTokenId,
```

As shown above, the amount to be withdrawn does not check that the amount is withdrawable when the token is an NFT. When the token is an ERC20, the amountOrTokenId variable is required to be greater than or equal to the maximum withdrawable amount.
Risk/Impact: *The core intended functionality of this contract does not work; authorized users can withdraw "locked" tokens at any time, and can additionally drain the contract of any tokens of the same type.*

Recommendation: *The withdrawable amount should be checked when the token is an NFT. In addition, the require statement should be updated to ensure that the amountOrTokenId is less than or equal to the maximum withdrawable amount.*
Resolution: *The team has implemented the above recommendation.*

OKLGTokenLocker - Finding #2 - High (Resolved)

Description: *The amount withdrawn from a locker is not properly updated when withdrawing tokens. Instead of adding to amountWithdrawn after a withdraw takes place, it is set to the amount that was withdrawn in that transaction only. In addition, this value is updated after the token transfer takes place, opening the door for reentrancy attacks if the token locked is an ERC-777 compliant token.*

```
_locker.amountWithdrawn = _locker.isNft ? 1 : _amountOrTokenId;
```

Risk/Impact: *Users will be able to withdraw an unlimited amount of tokens from the contract by dividing their withdraws into smaller amounts.*
Recommendation: *Instead of setting the amountWithdrawn to the amount withdrawn in the single transaction, it should be increased by the amount that was withdrawn. The token transfer should also occur after the amountWithdrawn is updated.*
Resolution: *The team has implemented the above recommendation.*

OKLetsApe - Finding #3 - High (Resolved)

Description: *The custom mint function can potentially cause issues with standard mints due to a token ID being reserved.*
Exploit Scenario:

1. *A token with an ID of 5 is custom minted.*
2. *Standard mints occur until the _tokenIds counter reaches 5.*
3. *Further minting will fail as the token ID of 5 already exists.*

Risk/Impact: *Standard minting will no longer function once the token counter has reached a token ID that has already been custom minted.*
Recommendation: *If the token ID counter reaches a token that already exists, it should be incremented. Disabling custom minting for specific token ids would also resolve this issue.*
Resolution: *The team has added logic to prevent any custom mints for any token ID that is greater than the current token ID counter, preventing this scenario from occurring.*

OKLGAirdropper - Finding #4 - High (Resolved)

Description: *The bulkSendMainTokens() function does not check if any blockchain's native token was passed into it (apart from the product cost payment) before sending the blockchain's native token to the list of addresses.*
Risk/Impact: *Any user has the ability to send the contract's native blockchain token balance to the list of addresses without providing funds.*
Recommendation: *The function should ensure that the total amount to send does not exceed the amount provided minus the Product cost. Additional functionality in the OKLGSpend contract will be necessary in order to fetch the*

Product cost in the blockchain's native token.
Update: *While the team has added logic to ensure that the user has sent enough funds to cover the total Airdrop amount, there is no check to ensure that they have sent enough funds for the Airdrop plus the Product cost. This allows the user to potentially use this contract's funds for the Airdrop.*
If the team does not wish to implement the above recommendation, they could also resolve this issue by requiring that the contract balance - msg.value before the Airdrop is less than or equal to the contract balance after the Airdrop.
Update #2: *The team has updated the code, however it does not make the correct balance checks to resolve this issue. The updated code will disable Airdrops in the blockchain's native token. The previous recommendation can be implemented to resolve the issue by updating the initialization of "balanceBefore" to the following:*

```
uint256 balanceBefore = address(this).balance - msg.value;
```

Resolution: *The team has implemented the above recommendation.*

AtomicSwap - Finding #5 - High (Resolved)

Description: *The setOracleAddress() function executes a function call for all of the contracts it manages.*
Risk/Impact: *If the list of contracts that the AtomicSwap contract manages is too large, the gas usage may exceed its limit. As the size of the contract list increases as more users purchase the product, the setOracleAddress functionality may break.*
Recommendation: *The number of contracts that the setOracleAddress() function calls should be limited.*
Update: *The team has added "_start" and "_max" parameters along with the following code:*

```
uint256 _index = 0;
uint256 _numLoops = _max > 0 ? _max : 50;
while (_index + _start < _start + _numLoops) {
  OKLGAtomicSwapInstance _contract = OKLGAtomicSwapInstance(
    targetSwapContracts[_index].sourceContract
  );
  _contract.setOracleAddress(oracleAddress);
  _index++;
}
```

As shown above, the contracts will always be updated from index 0. This will prevent contracts of larger indexes from being updated if the targetSwapContracts array becomes too large. The above while loop should instead begin at the passed "_start" index and continue until it has exceeded (_start + _numLoops).
Update #2: *The team has updated the code to where the beginning index is set to the passed "_start" index, however it is not incremented, resulting in the same contract being updated in each iteration. The team should update the following line inside the "while" loop to the following to resolve this issue:*

```
OKLGAtomicSwapInstance _contract = OKLGAtomicSwapInstance(
   targetSwapContracts[_start + _index].sourceContract
```

Resolution: *The team has implemented the above recommendation.*

OKLGFaasToken - Finding #6 - High (Resolved)

Description: *Rewards are calculated based on the user's OKLGFaasToken balance.*

Risk/Impact: *A user can claim their rewards, transfer their funds to another address, then claim the rewards again. This can be repeated, allowing an unlimited amount of rewards to be claimed.*
Recommendation: *Staked amounts should be stored in a mapping tied to a user's address. Rewards can then be calculated based on this amount.*
Resolution: *Users staked amounts are now additionally tracked using a mapping which is updated only when a user leaves or enters staking, preventing this exploit from occurring. Users should now be careful to not transfer their OKLGFaaSTokens or they can risk losing their staked funds or rewards.*

OKLG - Finding #7 - Medium

Description: *While the setFeeSellMultiplier() function requires that the sum of the fees is below 40, the individual fee setters do not, which can be used to bypass the limit.*
Risk/Impact: *Buy and sell fees can be set to an unlimited percentage, potentially resulting in a larger fee than expected.*
Reproduction Steps:

1. *Each individual fee is set to 0.*
2. *The fee multiplier is set to 5. As the total fees will be less than 40% with this multiplier, this call is permitted.*
3. *Each individual fee is then set to 10. The total fees with the fee multiplier of 5 is now over 100%.*

Recommendation: *Each individual fee setter should check that the total resulting fees do not exceed 40%.*

OKLGPasswordManager - Finding #8 - Medium

Description: *The price for a single store and a bulk store are currently the same even though their prices are intended to be different.*
Risk/Impact: *Users will be able to pay the same price for a single store or multiple stores.*
Recommendation: *If functionality to fetch this Product's price is implemented, a check can be made to ensure the provided amount is at least the calculated price based on the number of Accounts they are storing. The intended 50% discount can be included in this calculation.*

OKLGPasswordManager - Finding #9 - Medium (Partially Resolved)

Description: *Users can store multiple Accounts with the same ID.*
Risk/Impact: *If a second Account with the same ID is stored, users will not be able to update or delete the Account. In addition, users will not be able to fetch the second Account by its ID.*
Recommendation: *Disallow users from storing an Account with an ID equal to one that they have already stored.*
Update: *The team has implemented the following lines in the addAccount() function:*

```
require(
  userAccountIdIndexes[msg.sender][_id] == 0,
  'this ID is already being used, the account should be updated instead'
);
userAccountIdIndexes[msg.sender][_id] = userAccounts[msg.sender].length;
```

While this partially resolves the issue, users will still be able to overwrite their first created Account. A user's first created Account will set userAccountIdIndexes[msg.sender][_id] to 0 as their Accounts length is 0. If a user tries to

create another Account with this same ID, the above require statement will pass, and the user's first Account will be overwritten.

OKLGRaffle - Finding #10 - Low

Description: *In the drawWinner() function, all the information used in the calculation to draw the winning ticket is on-chain, This is common, albeit not best practice. Miners and bots in the memory pool may be able to predict the results and may take action accordingly to secure profits; though the chance of this is extremely low.*
Risk/Impact: *There is a chance that a user can wait to draw the winner until they have predicted the desired winning address. As a winning ticket cannot be drawn until the raffle has ended and anyone can call the function to draw the winner, the likelihood of this is extremely low, but is slightly higher in Raffles with a low number of participants.*
Recommendation: *Chainlink can be used as a reliable and secure source of randomness that cannot be manipulated or predicted by miners.*

OKLGPasswordManager - Finding #11 - Low (Resolved)

Description: *In multiple instances, the _userInfo array must be looped through in order to find the Account corresponding to a specified ID.*
Risk/Impact: *While this is not a major issue unless a user has stored a massive number of Accounts, it is a waste of gas.*
Recommendation: *The userAccounts mapping should be converted into a mapping(address => mapping(ID => AccountInfo[])). An Account could then be found in O(1) time by using the ID as a key instead of looping through their Accounts array.*
Resolution: *The team has implemented a second mapping to track the indexes of each user's Account IDs. While this is a less efficient solution than proposed, a user's Account list no longer has to be looped though to find a specific Account.*

OKLGSpend - Finding #12 - Low

Description: *Any excess funds sent to this contract as payment for a Product will not be returned to the user purchasing the Product.*
Risk/Impact: *Users will lose any excess funds sent as payment.*
Recommendation: *Excess funds should be transferred back to the corresponding OKLGProduct contract, where it should then be transferred back the the user.*
Update: *Excess funds are now returned back to the OKLGProduct it was sent from; however, these funds are not subsequently returned to the user.*

KetherNFTLoaner - Finding #13 - Low

Description: *If a user removes a loan, any excess amount sent in the blockchain's native token sent is not returned to them after the loanee is paid. Excess funds are instead stored in the contract.*
Risk/Impact: *Users will lose any excess funds sent as payment.*

Recommendation: *Excess funds should be calculated and transferred back to the sender.*

OKLG - Finding #14 - Informational

Description:. *The current buyback amount is calculated based on a percentage the amount of tokens in the OKLG token's Uniswap Pair address.*
Risk/Impact: *If the buyback token is not set to OKLG, it is possible that the buyback amount can cause the buyback token to be susceptible to frontrunning.*
Recommendation: *The project team should ensure that the buyback amount is not too large to mitigate the risk of frontrunning.*

CONTRACTS OVERVIEW

KetherNFTLoaner Contract:

- *This contract allows users to loan their KetherNFTs to other users for a specified time and cost.*
- *Users can add a "plot" at any time, transferring their specified KetherNFT to this contract and allowing users to purchase it as a loan, given it is not currently loaned out.*
- *Users must pay a "loan service charge" in order to add a plot.*
- *When adding, a user can choose to specify a loan price per day in the blockchain's native token and a max loan duration for their KetherNFT.*
- *The user can update these values to any amount at any time.*
- *If the values are not specified, they will default to the contract's default prices and max durations.*
- *After a plot has been added, users can borrow the plot for any amount of time less than or equal to the loan's maximum loan duration by paying the appropriate amount.*
- *A percentage fee is taken from the total payment amount and transferred to the owner of this contract; the remainder is transferred to the plot owner.*
- *After payment is sent, the KetherNFT's publish() function is called using the loanee's specified publish parameters.*
- *The loanee can republish the KetherNFT with different publish parameters at any time while their loan is still active.*
- *If a plot's loan is inactive, the plot owner can publish the KetherNFT.*
- *As the KetherNFT contract was not in the scope of this audit, we cannot verify the security or functionality of this contract's interactions with it.*
- *Once a loan has ended, the plot owner must remove the loan.*
- *If a plot owner wishes to cancel a loan early, they can pay the original loan price back to the loanee.*
- *The loan service charge, default loan price per day, default max loan duration, and loan percentage fee values can be updated by the owner at any time.*
- *Once a KetherNFT has been transferred to this contract, it cannot be transferred back to its original owner.*

MTGYOKLGSwap Contract:

- *This contract allows users to swap The Moontography Project token ($MTGY) for $OKLG at a defined exchange ratio.*
- *The owner can update the exchange ratio of $MTGY to $OKLG at any time.*
- *The owner can withdraw any tokens from this address at any time.*

- *This contract must be funded with OKLG tokens in order to complete swaps.*

OKLetsApe Contract:

- *The maximum supply of the ok.lets.ape. NFT ($OKLApe) is 10,000.*
- *Users can choose to burn their tokens in order to reduce the total supply, if desired.*
- *Users can pay a specified price in the blockchain's native token in order to mint tokens.*
- *The minting price is then forwarded to a payment address, which can be updated by the owner at any time.*
- *Any extra of the blockchain's native token sent to the contract when minting will be returned to the user.*
- *Users can hold up to a "maximum wallet amount" of tokens in their address at any time.*
- *Users can only mint tokens if a public sale is active, or if a presale is active and they have been added to the whitelist.*
- *In each public or presale round, only a limited amount of tokens can be minted.*
- *The owner can reset the round's mint counter or update the maximum mints per round at any time.*
- *If a "mint cost contract" is set, the price to mint a token will be fetched from that contract address.*
- *If the mint cost contract is not set, the price will be set to this contract's specified mint cost.*
- *The owner can update the mint cost contract address or this contract's mint cost amount at any time.*
- *This contract also supports the EIP-2981 NFT royalty standard.*
- *The owner can update the royalty address and the royalty percentage at any time.*
- *The owner can start and stop a presale or public sale round at at any time.*
- *The owner and authorized users can mint any amount of tokens for free, as long as it does not result in the token supply exceeding the maximum supply.*
- *The owner and authorized users can also mint using any token ID that has been previously burned.*
- *The owner and authorized users are exempt from the maximum wallet amount limitation.*
- *The owner can add or remove any address from the presale whitelist at any time.*
- *The owner can grant or revoke authorization to any address at any time.*
- *The owner can update the maximum wallet amount and the base URI for tokens at any time.*
- *The owner can pause the contract at any time, disallowing token transfers.*

OKLG Contract:

- *OKLG is a community-driven ERC20 token with a total supply of 420690000000.*
- *All tokens are sent to the owner upon deployment. Trading begins as disabled, preventing buys and sells for non-excluded addresses. Excluded addresses are only permitted to buy tokens during this time.*
- *Once trading is enabled by the owner, it cannot be disabled.*
- *In addition to the trading limitation, the owner can disable transfers of any kind for every address except transfers from themselves at any time.*
- *Excluded addresses can still purchase tokens while trading is disabled.*
- *When buying or selling, a percentage tax fee and project fee are taken.*
- *The tokens collected through the tax fee are removed from the circulating supply; this serves as a frictionless fee redistribution which automatically benefits all token holders at the time of each transaction.*
- *On each sell, a capped amount of accumulated project fees are swapped for the blockchain's native token. Percentages of this amount are then sent to a treasury address, used for the blockchain's native token rewards, and used to buy a specified "buyback token" which is then sent to a buyback address.*
- *The capped amount to swap is calculated using a percentage of the Uniswap Pair's token balance. This percentage can be updated by the owner to any amount at any time.*
- *The project team exercise caution when setting this percentage to a large amount due to risk of frontrunning.*
- *The owner can update the tax fee to maximum of 10% at any time.*

- *The owner can update the project fee to any percentage at any time.*
- *The owner can update the distribution of the project fee to any percentages at any time.*
- *The owner can exclude any address from transfer fees and dividends at any time.*
- *Users can also be excluded from fees through an owner-specified Fee Exclusion contract.*
- *The Fee Exclusion contract, treasury address, buyback token address, buyback receiver address, can be updated by the owner at any time.*
- *Users can claim rewards in the blockchain's native token or other token rewards based on the users OKLG balance.*
- *Other tokens sent to this contract can also be claimed as rewards based on the user's OKLG balance.*
- *When claiming, a user can earn additional rewards if they are determined to be eligible for a reward booster by a Boost Rewards contract.*
- *The rewards booster percentage can be updated by the owner at any time.*
- *Both the Fee Exclusion and Boost Rewards contract were not included in the scope of this audit, so we are unable to provide an assessment of this contract with regards to security.*
- *If a user either claims rewards or buys, sells, or receives tokens, a claim wait period must pass before they can claim rewards again.*
- *The owner can update the claim wait to any amount at any time.*
- *The owner can withdraw any of the blockchain's native token from the contract at any time.*
- *The owner can add or remove any address from the list of Uniswap Pair addresses at any time.*
- *The owner can add or remove any address from the blacklist at any time, disabling any transfers to or from the address.*
- *Any address that purchases tokens in the same block that trading is enabled will be automatically added to the blacklist.*

OKLGSpend Contract:

- *This contract is used to manage Product prices and ensure that payments for Products are sufficient.*
- *When payment for a Product is sent to this contract, it ensures that the required amount has been sent according to the Product's price. The Product price is then sent to a payment wallet.*
- *Any excess payment sent will be returned to the user.*
- *Prices are converted from the blockchain's native token to USD using a PriceFeed contract.*
- *The PriceFeed contract was not provided in the scope of this audit, so we are unable to provide an assessment of this contract with regards to security.*
- *The owner can override product prices for specified addresses at any time.*
- *The owner can update the paymentWallet and PriceFeed contract at any time.*
- *The owner can also update the price of a Product at any time.*
- *The owner can withdraw any of the blockchain's native token or other tokens from this contract at any time.*

OKLGProduct Contract:

- *This contract is inherited by the specified contracts below.*
- *When Products are paid for, the amount is sent to the OKLGSpend contract where the Product's price is calculated and subsequently sent to a payment address.*
- *The owner can update the OKLGSpend contract address, the Product's ID, and a referenced token address at any time.*
- *The owner can withdraw any of the blockchain's native token or other tokens from this contract at any time.*

OKLGAirDropper Contract:

- *This contract is an OKLGProduct that allows users to send any ERC20, ERC721, or the blockchain's native token amounts to a list of addresses.*
- *To send an airdrop, users must pay a product fee in the blockchain's native token along with providing the tokens to be airdropped.*
- *Airdrops of the blockchain's native token currently do not work as intended; users will not be able to send these Airdrops.*

OKLGAtomicSwapInstance Contract:

- *This contract is an OKLGProduct that allows users to swap a specified token between chains using an instance of this contract on each chain.*
- *To swap, users must pay a product fee in the blockchain's native token to create the contract; this amount is then sent to the OKLGSpend contract.*
- *If a max swap amount has been set, users cannot exceed it in a single swap.*
- *A gas amount is then sent from this contract to an Oracle address; users should ensure that this contract has a sufficient balance before initiating a swap.*
- *A swap ID will be created using a hash of the user, current timestamp, and the amount passed.*
- *The user must then pass the generated swap ID, swap timestamp, swap amount, and an additional gas fee to the instance of this contract on chain they are swapping to. The gas fee is then transferred to the Oracle address.*
- *Users cannot receive funds from the swap until the owner of the contract triggers the function to either refund the user on the original chain, or send the user tokens on the desired chain.*
- *If a token has a different number of decimals on each chain, the converted amount will be properly calculated using "target decimals" value, which should be set to the token's decimals on the other chain.*
- *The owner can update the target decimals to any amount at any time.*
- *As a fake swap ID and parameters can be passed into the contract, the owner must ensure that the user had actually initiated the swap on the original chain.*
- *In addition, the owner should be sure to remove the user's swap or mark it as complete on the chain which was not used to send tokens to the user.*
- *As these owner responsibilities require the use of off-chain logic, we are unable to provide an assessment of this contract with regards to security or proper functionality.*
- *The owner can update the gas fee to any amount at any time.*
- *The owner or a specified "token owner" address can set the contact to Active or Inactive at any time, enabling or disabling users from initiating a swap, receiving refunds, or completing a swap.*
- *Note that if a user has deposited their funds into an AtomicSwapInstance, and the instance on the chain to withdraw from is either Deactivated or set to Deactivated after the funds are deposited, users will not be able to complete their swap.*
- *The token owner can withdraw any of the contract's supported token at any time.*
- *The token owner address can update itself at any time.*
- *The owner can mark any swap as complete or incomplete at any time, either allowing a user to receive tokens multiple times or preventing them from receiving their tokens.*
- *The owner can update the oracle address, minimum gas for operation value, and decimals used to account for decimal differences when swapping.*
- *This contract should not be used with ERC-777 tokens.*

OKLGAtomicSwap Contract:

- *This contract is an OKLGProduct that is used to manage and create new OKLGAtomicSwapInstance contracts*

with specified attributes, including an amount of the specified token to be transferred to the contract.

- *Users must pay a product fee in the blockchain's native token to create the contract, which is sent to the OKLGSpend contract.*
- *When creating an OKLGAtomicSwapInstance, the user should ensure that a significant percentage of the total supply is stored in the contract so that funds exist when swapping occurs.*
- *Users should exercise caution when using fee-on-transfer tokens (unless the proper exemptions are made).*
- *The token owner of the newly created OKLGAtomicSwapInstance is set to the creator, and ownership is transferred to an Oracle address.*
- *Attributes of the OKLGAtomicSwapInstance to be deployed on the target chain are also stored in this contract.*
- *OKLGAtomicSwapInstances created through this contract can have their attributes updated by the owner, creator, or the oracle address at any time.*
- *The owner can update the product fee to any amount at any time.*
- *The owner can update the oracle address for each created AtomicSwapInstance at any time.*
- *As the Oracle address was not included in the scope of this audit, we are unable to provide an assessment of this contract with regards to security or proper functionality.*

OKLGFaasToken Contract:

- *Any user may stake a specified ERC20 or ERC721 token into this contract's pool to earn rewards in a specified reward token.*
- *Staked tokens will be locked in this contract until their stake time has passed or the contract's unlock date has passed.*
- *Upon staking, an equivalent amount of OKLGFaaSTokens will be minted to the user. Their amount staked will also be stored in the contract.*
- *Users should not transfer their OKLGFaaSTokens to different addresses or they can potentially lose their staked funds and rewards.*
- *Users will earn a reward amount on each block based on the amount staked and the pool's reward rate.*
- *When unstaking, the user's OKLGFaaSTokens are transferred from them to the 0x..dead address.*
- *An emergency unstake function also exists, which will send the user all of their deposited tokens without claiming any rewards.*
- *Rewards are claimed when both staking and unstaking. Users can also harvest rewards without a deposit or withdrawal.*
- *If the reward token is equal to the stake token, users can choose to automatically stake earned rewards back into this contract when harvesting.*
- *Automatic compounding functionality is only supported for ERC20 tokens.*
- *Users will not be able to stake tokens or earn further rewards after the pool's end block has been reached.*
- *The end block is calculated using the pool's reward token's base supply divided by the pool's rewards per block.*
- *The creator of the pool has the ability to update the pool's base supply or current supply at any time.*
- *The creator should ensure that the contract's reward token balance is sufficient to distribute rewards.*
- *The creator or token owner address of the contract can withdraw the contract's reward token balance to the token owner address at any time.*
- *If this occurs, users will be able to withdraw their staked tokens from the contract.*
- *The team should avoid using ERC-777 tokens as the contract's reward token.*
- *The token owner can withdraw all rewards from the contract at any time. If this occurs, users will be able to withdraw their staked tokens.*

OKLGFaaS Contract:

- *This contract is an OKLGProduct that allows users to create an OKLGFaasToken contract with specified attributes.*
- *Users must pay a product fee in the blockchain's native token to create the contract, which is sent to the OKLGSpend contract.*
- *When a new OKLGFaaSToken contract is created, the creator address is set to this contract and the token owner address is set to the user.*
- *If an OKLGFaaSToken contract is created using this contract, its pool's base supply and current supply cannot be updated after contract creation.*
- *The user can "remove" their contract if the contract's lock time has passed, which will withdraw the contract's reward balance to them and set the contract's reward supply to 0.*

OKLGRaffle Contract:

- *This contract is an OKLGProduct that allows users to create Raffles with a reward token and amount/id, ticket token and price, start and end time, and maximum number of entries.*
- *Users must pay a product fee in the blockchain's native token to create a Raffle, which is sent to the OKLGSpend contract.*
- *Users can offer either an ERC20 token reward or an NFT as the Raffle's reward.*
- *Raffle participants can purchase tickets by passing in the ID of the raffle and the number of tickets to purchase.*
- *Participants can only purchase tickets for a Raffle between its start and end times.*
- *The total tickets a participant can purchase cannot exceed a specified maximum ticket amount per address unless this limit has not been set.*
- *Once the Raffle end time has passed, any address can trigger the function to draw the winner.*
- *A percentage administration fee is taken from the accumulated tokens from ticket sales and sent to this contract's owner; the remainder will be sent to the Raffle's owner.*
- *A random number corresponding to a ticket is then drawn by hashing the number of tickets and current block attributes.*
- *As all the information used in the calculation is from chain, miners and bots in the memory pool may be able to predict the results and may take action accordingly to secure profits; though the chance of this is extremely low. Chainlink can be used as a source of randomness that cannot be manipulated or predicted by miners.*
- *The ERC20 tokens or NFT is then transferred to the owner of the winning ticket, and the Raffle is marked as complete.*
- *If a Raffle has not been completed, its owner can choose to close and refund it, transferring the appropriate funds back to each user.*
- *If a Raffle is refunded, it will be marked as Completed.*
- *The owner of the contract can update the administration fee to any amount at any time.*
- *The owner of a Raffle should make sure they set an end date or else the winner can be drawn at any time.*
- *The owner of a Raffle can update the Raffle's end time and transfer its ownership as long as the Raffle has not been completed.*
- *This contract should not be used with ERC-777 tokens.*

OKLGPasswordManager Contract:

- *This contract is an OKLGProduct which allows users to store information in Accounts.*
- *Users must pay a product fee in the blockchain's native token to create an Account, which is sent to the OKLGSpend contract.*
- *Users can also pay the same product fee to create multiple accounts at once.*
- *When creating an Account, a user will specify three strings: an "id", "iv", and "ciphertext".*

- *Users should be careful to not create an Account with the same id as their first created Account, as users will not be able to update or delete the first Account or fetch the Account by ID.*
- *A corresponding Account will be created which simply stores these strings along with its created timestamp and a marker to check if it has been marked as deleted.*
- *A user can update an account's iv and ciphertext data at any time at no cost.*
- *A user can "delete" any of their Accounts at any time, which will update the Account's timestamp and mark it as deleted.*
- *Users will not be able to create another Account with the same ID as a deleted Account (unless it was their first Account created).*
- *The Account will still remain stored in the contract if "deleted".*

MTGYTokenLocker Contract:

- *This contract is an OKLGProduct that allows users to create Token Lockers.*
- *Users must pay a product fee in the blockchain's native token to create a Token Locker, which is sent to the OKLGSpend contract.*
- *When creating a Token Locker, users will specify the token to be locked, an end time, vesting periods, and authorized addresses that can withdraw the vested tokens.*
- *The number of vests will be distributed evenly across the total vesting period, where a linear amount of vested tokens can be withdrawn.*
- *The specified tokens will be transferred from the locker owner to this contract.*
- *If the number of vests is set to 1, all tokens will not be withdrawable until the end of the lock time.*
- *If an NFT is used as the locked token, it will also not be withdrawable until the end of the lock time.*
- *The owner of a locker can transfer its locker ownership at any time.*
- *The owner of a locker can update the locker's end time at any time.*

OKLGRewards Contract:

- *OKLG token holders can use this contract to claim rewards in the blockchain's native token or in reward tokens, based on the amount of OKLG they hold.*
- *Users must wait until both this contract's claim wait has expired, and their OKLG claim wait has expired before claiming rewards.*
- *If a user is determined to be eligible by an associated Booster contract, users can earn additional rewards based on an associated BoostRewardsMultiplier contract's multiplier. If there is no BoostRewardsMultiplier contract set, this contract's multiplier will be used.*
- *The owner can reset any user's last claim time at any time, allowing them to be eligible to receive rewards instantly.*
- *The owner can update the OKLG contract, Booster contract, BoosterRewardsMultiplier contract, claim wait, and reward multiplier at any time.*
- *The owner can withdraw any of the blockchain's native token or any other token from this contract at any time.*

OKLGTrustedTimestamping Contract:

- *This contract is an OKLGProduct that allows users to store Data Hashes.*
- *Users must pay a product fee in the blockchain's native token to store a hash, which is sent to the OKLGSpend contract.*
- *When storing a hash, users will specify a hash, file name, and file size in bytes, which are stored in the Data Hash.*
- *The DataHash will then be created, which consists of the specified attributes with the user attributes, along with*

the current timestamp.

- *Users can pass an address into this contract in order to fetch its stored DataHashes.*
- *Users can also fetch what address a DataHash belongs to by passing in its hash.*

EXTERNAL THREAT RESULTS

Vulnerability Category	Notes	Result
Arbitrary Storage Write	N/A	PASS
Arbitrary Jump	N/A	PASS
Centralization of Control	The owner has the permissions listed above. The owner of any contract that is an OKLGProduct can withdraw any funds from it at any time. The owner or an Admin can withdraw any amount of funds from the AssetManager contract. The OKLGAtomicSwapInstance contract requires the use of off-chain logic to function properly. The owner of the OKLetsApe contract can mint tokens for free at any time.	WARNING
Delegate Call to Untrusted Contract	N/A	PASS
Dependence on Predictable Variables	It is possible for the Raffle winner to be predicted; however, the likelihood of this along with its use for exploitation is very low.	PASS
Deprecated Opcodes	N/A	PASS
Ether Thief	N/A	PASS
Exceptions	N/A	PASS
External Calls	N/A	PASS
Flash Loans	N/A	PASS
Frontrunning	If the buyback token is not set OKLG token in the OKLG Contract, and the buyback token may be susceptible to frontrunning.	WARNING
Integer Over/Underflow	N/A	PASS
Logical Issues	Issues mentioned above.	WARNING
Multiple Sends	N/A	PASS
Oracles	N/A	PASS

Vulnerability Category	Notes	Result
Suicide	N/A	PASS
State Change External Calls	N/A	PASS
Unbounded Loop	N/A	PASS
Unchecked Retval	N/A	PASS
User Supplied Assertion	N/A	PASS
Critical Solidity Compiler	N/A	PASS
Overall Contract Safety		WARNING

CONTRACT SOURCE SUMMARY AND VISUALIZATIONS

Name	Address/Source Code	Visualized (Hover-Zoom Recommended)
HasERC20Balance	GitHub	Function Graph. Inheritance Chart.
HasERC721Balance	GitHub	Function Graph. Inheritance Chart.
IsERC20HODLer	GitHub	Function Graph. Inheritance Chart.
KetherNFTLoaner	GitHub	Function Graph. Inheritance Chart.
MTGYOKLGSwap	GitHub	Function Graph. Inheritance Chart.
OKLetsApe	GitHub	Function Graph. Inheritance Chart.
OKLG	GitHub	Function Graph. Inheritance Chart.
OKLGAirdropper	GitHub	Function Graph. Inheritance Chart.

OKLGAtomicSwap	GitHub	Function Graph. Inheritance Chart.
OKLGAtomicSwapInstHash	GitHub	Function Graph. Inheritance Chart.
OKLGFaaS	GitHub	Function Graph. Inheritance Chart.
OKLGPasswordManager	GitHub	Function Graph. Inheritance Chart.
OKLGRaffle	GitHub	Function Graph. Inheritance Chart.
OKLGRewards	GitHub	Function Graph. Inheritance Chart.
OKLGSpend	GitHub	Function Graph. Inheritance Chart.
OKLGTokenLocker	GitHub	Function Graph. Inheritance Chart.
OKLGTrustedTimestamping	GitHub	Function Graph. Inheritance Chart.

GO HOME