#HashEx
BLOCKCHAIN SECURITY

Uniqo

smart contracts final audit report

October 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Uniqo team to perform an audit of their smart contract. The audit was conducted between 05/09/2022 and 09/09/2022.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The audited contract is designed to be deployed with proxies. Users have no choice but to trust the owners, who can update the contracts at their will.

The code was provided in UNIQO.sol file with 3866c9265b975f07bd55274731bb068e MD5 sum.

Update: the Uniqo team has responded to this report. The updated code has a78d3aedbb62617752394718ba695523 MD5 sum.

2.1 Summary

Project name	Uniqo
URL	https://uniqo.finance
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
Uniqo	

3. Found issues

High	1 (9%)
Medium	1 (9%)
Low	4 (36%)
Info	5 (46%)

C1. Uniqo

ID	Severity	Title	Status
C1-01	High	Exaggerated owners' rights	Partially fixed
C1-02	Medium	Standard violation	Resolved
C1-03	Low	Allowance can be increased during pauses	Resolved
C1-04	Low	Not using SafeERC20 for arbitrary tokens	Resolved
C1-05	Low	Typos	Resolved
C1-06	Low	Gas optimisation	Resolved
C1-07	Info	Real asset value is not changed by reward/ rebound mechanism	Acknowledged
C1-08	Info	initialize() should be modified during proxy upgrades	Acknowledged
C1-09	Info	clearStuckBalance() and receive() functions should be deleted	Resolved

C1-10	Info	Swaps don't have min amount and deadlines	Acknowledged
C1-11	Info	Pairs not tracked in automatedMarketMakerPairs[] mapping should not be used for liquidity adding	Acknowledged

4. Contracts

C1. Uniqo

Overview

An ERC20 token implementation with an adjustable total supply mechanism allowing to keep the token price almost constantly growing. The increase or decrease of the total supply trend is determined by token price in UNIQO/BUSD pair. If the price has dropped by five percent from the previous all-time high (ATH), the total supply is modified in a way new price exceeds the previous ATH by one percent (can be changed up to 100% of the previous ATH). All operations with the token are taxed except if a user transfers all their wallet assets to another not-pair address. Accumulated taxes are burned, sent to the treasury, and used for buyback proportionally to chosen percentages.

Issues

C1-01 Exaggerated owners' rights

High Partially fixed

If the owner's account is compromised or because of negligence, it may block token transfers or compromise users' funds.

a. The owner can `setBackingLPToken()` to a wrong pair address what will halt all token transfers when it's buyback time.

b. The owner can pause all token transfers.

c. The owner can increase transfer fees up to 100% or beyond with `setFees()`. Also, `feeDenominator` can be set to arbitrary value breaking fees math.

d. Since during buyback tokens are transferred to the contract via `autoLiquidityReceiver`, it's crucial the contract always has a sufficient allowance, or the tokens transfers will be halted. Changing `autoLiquidityReceiver` to a malicious one may cause transfers' halt.

e. Unreasonable new values in the `setRewardRate()` and `setRewardFrequency()` functions can lead to overflows and/or division errors in new total supply calculations and therefore halt token transfers.

f. All LP tokens received as a result of buyback/liquify are accumulated by `autoLiquidityReceiver` address. This makes `autoLiquidityReceiver` holder of a significant liquidity share. The users' assets in UNIQO token may lose value If all these LP tokens are burned and swapped on the second token in pair at once.

g. The owner can remove the pair from `automatedMarketMakerPairs[]` mapping during pending reward, increase `rewardRate`, and drain assets as described in C1-09. To not wait until an appropriate moment, the owner can change the pair with a simulated token price increase. Thus, the owner can withdraw all assets from any pair containing UNIQO as one of the tokens.

h. The owner can upgrade token contract behind the proxy and change the logic, halt the transfers, or perform any other malicious actions.

Recommendation

a, b, f, e, g, h – Remove the functionality or transfer ownership to `Timelock` contract with a minimum delay of at least 24 hours and MultiSig as admin. This won't stop the owner from possible rights abuse, but it will help users to be informed about upcoming changes.

c, e, g – Add input parameters validation.

d – Avoid `autoLiquidityReceiver` medium and swap tokens directly to the contract.

f – `autoLiquidityReceiver` address should be `Timelock` contract with a minimum delay of at least 24 hours. This won't stop the owner from possible rights abuse, but it may help users to be informed about upcoming changes.

Update

c. Safety checks have been implemented in the `setFees()` function.

e. Check on input data has been implemented into `setRewardRate()`, but we consider added constraints insufficient. Firstly, overflow can occur not only during the first iteration, this case is not covered. Moreover, absence of overflow doesn't save from drastical value increase and owner's right abuse scenario described in g clause.

```
function setRewardRate(uint256 rate, uint256 denominator) external onlyOwner {
    require(rewardRate > 0, "INVALID_REWARD_RATE");
    require(denominator > 0, "INVALID_REWARD_RATE_DENOMINATOR");
    uint256 supply = _totalSupply;
    // check overflow
    require(supply.mul(denominator.add(rate)).div(denominator) > supply,
"INVALID_REWARD_RATE_PARAMS");
    ...
  }
```

Regarding `rewardFrequency` added validation we recommend also limit it from below as too small values may cause gas block exceedance during reward and temporary token transfers halt.

```
function setRewardFrequency(uint256 valueInSeconds) external onlyOwner {
   require(valueInSeconds <= 1 days && valueInSeconds > 0, "INVALID_REWARD_FREQUENCY");
   rewardFrequency = valueInSeconds;
   emit SetRewardFrequency(valueInSeconds);
}
```

Team response

The team is going to implement a Timelock and transfer ownership of the token to Timelock contract, so in this resolution, the team will be focusing on addressing parameter validation issues.

C1-02 Standard violation

Medium Resolved

No event is emitted in `_basicTransfer()`, which violates the ERC-20 token standard.

Recommendation

We recommend conforming to the standard and emitting a `Transfer()` event with every transfer.

C1-03 Allowance can be increased during pauses

Low Resolved

`increaseAllowance()` is accessible during pauses, whereas `approve()` calls are reverted. Since `approve()` also can be used for allowance increase this produce certain inconsistency.

C1-04 Not using SafeERC20 for arbitrary tokens

Low Resolved

The `rescueToken()` function doesn't use `safeTransfer()` method from SafeERC20 OpenZeppelin library (or any other universal transfer method for a non-standard ERC20-like tokens) for an arbitrary third-party token. Some tokens may be trapped even with the presence of `rescueToken()` function.

C1-05 Typos

Low Resolved

Typos reduce the code's readability. Typos in 'liner', 'rabase', 'ater', 'y = ax+b'.

C1-06 Gas optimisation

Low Resolved

a. External calls to itself in L276,317-318,795-796,913-914.

b. Unnecessary reads from storage:

- `_allowedFragments[msg.sender][spender]` in `decreaseAllowance()` and in `increaseAllowance()`

- `router`, `treasury`, `BUSD` in `withdrawFeesToTreasury()`
- `buyFee`, `sellFee`, `transferFee`, `feeDenominator` in `setFees()`
- `_makerPairs.length` in `setAutomatedMarketMakerPair()`
- `maxHoldingPercentTakeProfitApplied` in `calculateTakeProfitFactor()`
- `_makerPairs.length` in `manualSync()`
- `_makerPairs.length` in `getLiquidityBacking()`
- `history.totalSoldAmountLast24h` in `_checkDailyTakeProfitAndUpdateSaleHistory()`
- `BUSD`, `autoLiquidityFeePercent`, `autoLiquidityReceiver`, `router` in `_swapBack()`
- `feeDenominator`, `_gonsPerFragment` in `_takeFee()`
- `_gonsPerFragment` in `_shouldApplyTransferFee()`
- `athPrice`, `_totalSupply`, `rewardRateDenominator`, `rewardRate`, `lastRewardTime`, `rewardFrequency` in `_reward()`
- `pair` in `_getTransactionType()`
- `pair`, `athPrice` in `_estimateReboundSupply()`

C1-07 Real asset value is not changed by reward/ rebound mechanism

● Info ⊘ Acknowledged

Total users' wallet value nominated in BUSD (or other valuable tokens) is not affected by total supply reward and rebound variation, as rebasing affects pair reserves proportionally to total supply adjustment.

C1-08 initialize() should be modified during proxy upgrades

● Info ⊘ Acknowledged

The audited realization of `initialize()` contains pair creation and deployer balance setting to total supply. These code parts should be excised in case of proxy implementation change.

C1-09 clearStuckBalance() and receive() functions should be deleted

● Info ✓ Resolved

There are several reasons why the functions are redundant and should be removed. Firstly, the only legitimate way native currency can appear on the contract's balance is `pair` change to

BNB/UNIQO which already relates to C1-01(a) issue. However, if the Uniqo team acknowledges the issue, the pair setting to BNB/UNIQO will halt token transfers as a swap in `_swapBack()` calls router method for token-token exchange, consequently, the pair can't be changed to a pair with unwrapped native currency.

```
  function _swapBack() internal swapping {
    ...
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = BUSD;
    router.swapExactTokensForTokensSupportingFeeOnTransferTokens(amountToSwap, 0, path,
autoLiquidityReceiver, block.timestamp);
    ...
  }
```

C1-10 Swaps don't have min amount and deadlines

● Info ✓ Acknowledged

Swap and liquidity addition inside `_swapBack()` are executed with zero slippage and `block.timestamp` deadline which does not affect the transaction's success. These parameters should be obtained off-chain, otherwise, a buyback operation is vulnerable to sandwich attacks.

Team response

The swap is performed with 100% slippage as using `minAmountOut` is not useful on-chain. If a swap is triggered by a user using a normal slippage, the transaction would revert. `block.timestamp` is used instead of a future deadline, as the transaction is built on-chain.

C1-11 Pairs not tracked in automatedMarketMakerPairs[] mapping should not be used for liquidity adding

● Info ✓ Acknowledged

The reserves of pairs with a positive value in the `automatedMarketMakerPairs` mapping are synced during `_reward()` to maintain the prices relevant. The assets of pairs whose reserves are not synced while rebasing can be drained as pairs would consider increased balance as

the result of the token transfer. Users should provide liquidity with UNIQO tokens only using pairs with a positive value in the `automatedMarketMakerPairs` mapping.

5. Conclusion

1 high, 1 medium, 4 low severity issues were found during the audit. 1 medium, 4 low issues were resolved in the update.

Users must understand that the protocol total supply adjustment mechanism doesn't affect the asset's real value, but rather maintains permanent growth on crypto monitoring charts.

The reviewed contract is extremely dependent on the owner’s account. The contract is designed to be deployed with proxies and contains multiple functions able to halt transfers. Users using the project have to trust the owner and that the owner's account is properly secured.

We strongly suggest adding functional tests for the contract to cover all edge cases.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY