HashEx
BLOCKCHAIN SECURITY

REAKTOR Token

smart contracts
preliminary audit report
for internal use only

February 2024

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 REAKTOR Token team to perform an audit of their smart contract. The audit was conducted between 20/02/2024 and 22/02/2024.

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 code is available in the Ethereum mainnet at 0xb970E14Df2161c0A2f32EBA35901f2446581b482.

2.1 Summary

Project name	REAKTOR Token
URL	https://reaktortoken.com
Platform	Ethereum
Language	Solidity
Centralization level	● High
Centralization risk	● High

2.2 Contracts

Name	Address
Ownable	
REAKTOR	

3. Project centralization risks

The contract has a single owner, who has access to the following governance functions that can be misused or abused:

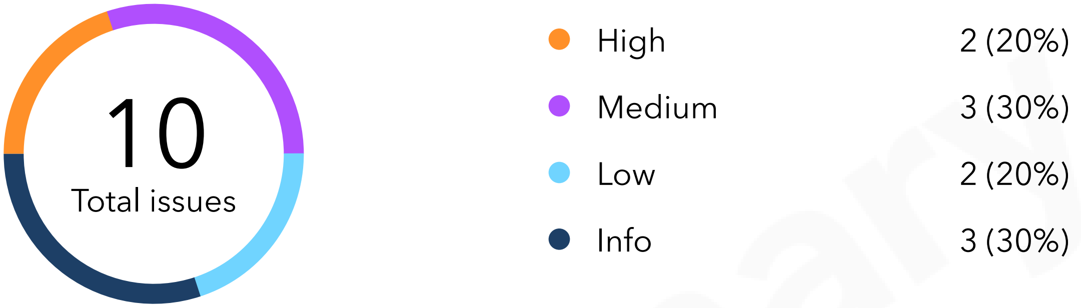
- the `excludeFromReward()` function can be used to increase `_excluded[]` array beyond reasonable length, possibly leading to a complete transfer blocking;

- token forced redistribution: the owner can exclude an account from the reward and include it back late, redistributing part of the tokens from holders in profit of the included account;

- the owner can set maximum token transfer amount to zero, effectively blocking most of the transfers;

- the owner controls fees, which can be set over 100% to revert all subjected token transfers.

4. Found issues



Ca4. Ownable

ID	Severity	Title	Status
Ca4Ib1	● High	Ownership transfer can be reverted	Open
Ca4Ib2	● Low	Misleading events	Open
Ca4Ib3	● Low	Inco	Open

Ca5. REAKTOR

ID	Severity	Title	Status
Ca5Ib4	● High	The excluded[] array length problem	Open
Ca5Ib8	● Medium	Locked ETH	Open
Ca5Ib5	● Medium	ERC20 standard violation	Open
Ca5Ib9	● Medium	Deterministic random	Open
Ca5Iba	● Info	Limited DeFi Protocol Compatibility	Open

Ca5Ib6	● Info	Lack of events on important changes	Open
Ca5Ib7	● Info	Swaps are made with 100% slippage and infinite deadline	Open

5. Contracts

Ca4. Ownable

Overview

The Ownable contract extends a simple single ownership model by implementing a temporary renouncing.

Issues

Ca4Ib1 Ownership transfer can be reverted ● High ? Open

The `unlock()` function is designed to regain ownership after a temporary renouncing. The problem is that it doesn't clear the `_previousOwner` variable, meaning the previous owner can reinstate himself as the owner after ownership been transferred or renounced.

```
/**
 * @dev Leaves the contract without owner. It will not be possible to call
 * `onlyOwner` functions anymore. Can only be called by the current owner.
 *
 * NOTE: Renouncing ownership will leave the contract without an owner,
 * thereby removing any functionality that is only available to the owner.
 */
function renounceOwnership() public virtual onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(
        newOwner != address(0),
        "Ownable: new owner is the zero address"
    );
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}

//Locks the contract for owner for the amount of time provided
function lock(uint256 time) public virtual onlyOwner {
    _previousOwner = _owner;
    _owner = address(0);
    _lockTime = now + time;
    emit OwnershipTransferred(_owner, address(0));
}

//Unlocks the contract for owner when _lockTime is exceeds
function unlock() public virtual {
    require(
        _previousOwner == msg.sender,
        "You don't have permission to unlock"
    );
    require(now > _lockTime, "Contract is locked until 7 days");
    emit OwnershipTransferred(_owner, _previousOwner);
    _owner = _previousOwner;
}
```

Recommendation

A custom contract may be used for REAKTOR token ownership, it should not be able to transfer ownership further or call the `unlock()` function.

Ca4Ib2 Misleading events

● Low ? Open

There are misleading events emitted in the `lock()` and `unlock()` functions: announce of ownership transfer should be replaced with a custom event for ownership locking.

```
    //Locks the contract for owner for the amount of time provided
    function lock(uint256 time) public virtual onlyOwner {
        ...
        emit OwnershipTransferred(_owner, address(0));
    }
    //Unlocks the contract for owner when _lockTime is exceeds
    function unlock() public virtual {
        ...
        emit OwnershipTransferred(_owner, _previousOwner);
    }
```

Ca4Ib3 Inco

● Low ? Open

The error message in the `unlock()` function states a fixed locking period of 7 days while actual lock time is an arbitrary number.

```
    //Locks the contract for owner for the amount of time provided
    function lock(uint256 time) public virtual onlyOwner {
        ...
        _lockTime = now + time;
    }

    //Unlocks the contract for owner when _lockTime is exceeds
    function unlock() public virtual {
        ...
        require(now > _lockTime, "Contract is locked until 7 days");
    }
```

Ca5. REAKTOR

Overview

A fork of the SafeMoon token - rebasing token based on the Reflect Finance model with added transfer fee to be added in UniswapV2 pair with wrapped ETH (WETH). The REAKTOR token implements a few minor updates: it has an internal randomizer to disable fees on approximately 25% of all transfers that would otherwise be subject to fees, it also has an additional onlyOwner function to precisely adjust max transfer amount.

Issues

Ca5Ib4 The excluded[] array length problem ● High ⓘ Open

The mechanism of removing addresses from auto-yielding implies a loop over excluded addresses for every transfer operation or balance inquiry. This may lead to extreme gas costs up to the block gas limit and may be avoided only by the owner restricting the number of excluded addresses. In an extreme situation with a large number of excluded addresses transaction gas may exceed the maximum block gas size and all transfers will be effectively blocked. If the owner's account gets compromised the attacker can make the token completely unusable for all users. Moreover, the function`includeInReward()` relies on the same `for()` loop which may lead to irreversible contract malfunction.

```
function excludeFromReward(address account) public onlyOwner {
    ...
    _excluded.push(account);
}

function includeInReward(address account) external onlyOwner {
    require(_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

Recommendation

The token owner should monitor gas usage of transfers when excluding new addresses. Also, the owner's account must be properly secured to mitigate the possibility of being hacked.

Ca5Ib8 Locked ETH ● Medium ⓘ Open

Part of collected fees is added as liquidity to the UniswapV2 pair without precise calculation of input amounts according to the current price. This leads to excessive ETH accumulated on the contract's balance without being able to transfer it out or add as liquidity in future `swapAndLiquify()` calls.

```
    function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
        // split the contract balance into halves
        uint256 half = contractTokenBalance.div(2);
        uint256 otherHalf = contractTokenBalance.sub(half);

        // capture the contract's current ETH balance.
        // this is so that we can capture exactly the amount of ETH that the
        // swap creates, and not make the liquidity event include any ETH that
        // has been manually sent to the contract
        uint256 initialBalance = address(this).balance;

        // swap tokens for ETH
        swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify
 is triggered

        // how much ETH did we just swap into?
        uint256 newBalance = address(this).balance.sub(initialBalance);

        // add liquidity to uniswap
        addLiquidity(otherHalf, newBalance);

        emit SwapAndLiquify(half, newBalance, otherHalf);
    }

    function swapTokensForEth(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();

        _approve(address(this), address(uniswapV2Router), tokenAmount);

        // make the swap
        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );
    }

    function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router), tokenAmount);

        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            owner(),
            block.timestamp
        );
    }
```

The contract also allows incoming ETH transfers.

Recommendation

The threshold for triggering the `swapAndLiquify()` call should be adjusted accordingly to the RKR/WETH UniswapV2 pair to mitigate excessive ETH due to pair's price impact.

Ca5Ib5 ERC20 standard violation ● Medium ? Open

Implementation of the `transfer()` function does not allow to input zero amount as it's demanded in ERC20 token standard. This issue may break the interaction with smart contracts that rely on full ERC20 support. Also, transfer functions of the reviewed contract don't throw error messages for the amounts bigger than the sender's balance (like `ERC20: transfer amount exceeds allowance` in OpenZeppelin's ERC20 implementation) which may confuse users.

```
function _transfer(address from, address to, uint256 amount) private {
    ...
    require(amount > 0, "Transfer amount must be greater than zero");
}
```

Recommendation

Any smart contracts to interact with the REAKTOR token must take this non-standard behavior.

Ca5Ib9 Deterministic random ● Medium ? Open

The `random()` function is designed to be called during all transfers between non-fee-exempt addresses. If the pseudo-random result is in low quarter, the transfer is exempt. The randomization relies on the on-chain data, and therefore, it cannot be considered as a truly secure random. A special router contract may be constructed to provide 100% guarantee for fee-free transfers.

```
function random() private view returns (uint) {
    return
        uint(
            keccak256(abi.encodePacked(block.difficulty, block.timestamp))
        );
}

function hasWonTheLottery() private view returns (bool) {
    if (_lotteryEnabled == false) {
        return false;
    }

    return random() % 100 < 25;
}

function _transfer(address from, address to, uint256 amount) private {
    ...
    //if any account belongs to _isExcludedFromFee account then remove the fee
    if (
        _isExcludedFromFee[from] ||
        _isExcludedFromFee[to] ||
        hasWonTheLottery()
    ) {
        takeFee = false;
    }
    ...
}
```

Recommendation

Disable the lottery completely.

Ca5Iba Limited DeFi Protocol Compatibility ● Info ⓘ Open

The REACTOR token is a rebase token with commissions on transfer. Some DeFi protocols may not support such tokens. For example, Uniswap V3 does not support such tokens. Prior to incorporating it into any DeFi protocol, exhaustive testing is imperative.

Ca5Ib6 Lack of events on important changes ● Info ⓘ Open

The functions `excludeFromFee()`, `includeInFee()`, `setTaxFeePercent()`, `setLiquidityFeePercent()`, `setMaxTxPercent()`, `setMaxTxAmount()`, `setLotteryEnabled()` change important state variables but don't emit any events.

Ca5Ib7 Swaps are made with 100% slippage and infinite deadline

● Info ⓘ Open

Swapping and adding to liquidity in the token contract are made with 100% slippage and an infinite deadline.

```
function swapTokensForEth(uint256 tokenAmount) private {
    // generate the uniswap pair path of token -> weth
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();

    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // make the swap
    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(this),
        block.timestamp
    );
}

function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // add the liquidity
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        owner(),
        block.timestamp
    );
}
```

6. Conclusion

2 high, 3 medium, 2 low severity issues were found during the audit. No issues were resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter.

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. Issue status description

Resolved. The issue has been completely fixed.

Partially fixed. Parts of the issue have been fixed but the issue is not completely resolved.

Acknowledged. The team has been notified of the issue, no action has been taken.

Open. The issue remains unresolved.

Appendix C. 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

Appendix D. Centralization risks classification

Centralization level

- **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

✉ contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx

BLOCKCHAIN SECURITY