#HashEx
BLOCKCHAIN SECURITY

Yetalaunch

smart contracts
final audit report

November 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 Yetalaunch team to perform an audit of their smart contract. The audit was conducted between 06/10/2022 and 12/10/2022.

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 code was provided in Yeta.sol file with 7587de5158043a8a65759240aeca289e MD5 sum.

Update: the Yetalaunch team has responded to this updated report. The updated Yeta.sol file has 627f4eee4c92b3f489f716f7e6d5bdc3 MD5 sum.

2.1 Summary

Project name	Yetalaunch
URL	https://yetalaunch.com/
Platform	Binance Smart Chain, Ethereum
Language	Solidity

2.2 Contracts

Name	Address
YetaLaunchpad	

3. Found issues

19
Total issues

●	High	7 (37%)
●	Medium	3 (16%)
●	Low	7 (37%)
●	Info	2 (10%)

C1. YetaLaunchpad

ID	Severity	Title	Status
C1-01	● High	Access to withdraw() function	✓ Resolved
C1-02	● High	Access to change tokens	✓ Resolved
C1-03	● High	Double call of the withdraw() function	✓ Resolved
C1-04	● High	Staking rewards are not guaranteed	✓ Acknowledged
C1-05	● High	Getting back tokens when removing IDO	✓ Resolved
C1-06	● High	DoS with block gas limit function removeIDO()	✓ Resolved
C1-07	● High	Access to change weights	✓ Acknowledged
C1-08	● Medium	Elements are not popped from the array	✓ Resolved
C1-09	● Medium	DoS with block gas limit	✓ Partially fixed
C1-10	● Medium	Timestamp may be out of time intervals	✓ Acknowledged

C1-11	● Low	Default variables visibility	Partially fixed
C1-12	● Low	No events	Partially fixed
C1-13	● Low	Unnecessary SafeMath	✓ Resolved
C1-14	● Low	No required function	✓ Acknowledged
C1-15	● Low	Lack of validation in addIDO()	✓ Resolved
C1-16	● Low	Gas optimization	Partially fixed
C1-17	● Low	No revert messages	✓ Acknowledged
C1-18	● Info	Custom ownable functionality	✓ Acknowledged
C1-19	● Info	No support of tokens with commissions	✓ Acknowledged

4. Contracts

C1. YetaLaunchpad

Overview

The launchpad contract for ERC20 tokens with limitations on the available tokens amount to purchase. A user's limit for a specific token depends on the project type, sale stage, and user's staked tokens amount. Bought tokens are not instantly sent to the acquirer, but are vested according to a project's schedule after sales end.

Issues

C1-01 Access to withdraw() function ● High ✓ Resolved

When a project's sale ends the raised funds and unsold tokens can be transferred out of the contract with the `withdraw()` function. The assets receiver is passed as a call parameter. As the function has an `onlyOwner()` modifier, the owner alone decides who gets the raised funds and the receiver of the left tokens. This right can be abused.

Recommendation

Add the address that will receive `rasingToken` from an IDO to add the possibility of checking this parameter before the IDO to the `Projects` struct.

Also, add the possibility to call this function by anyone.

C1-02 Access to change tokens ● High ✓ Resolved

The owner can change the addresses of `rasingToken` and `stakingToken` to malicious ones. This may lead to losing funds of stakers and participants of IDOs. Moreover, substituting `rasingToken` the owner can buy allocated tokens for nothing and then return the token address to the initial.

Recommendation

Delete the functions `setRaiseToken()` and `setStakeToken()`. In practice, they can't be changed safely because the contract must have zero stakes and zero IDOs to make it safely.

C1-03 Double call of the withdraw() function ● High ✓ Resolved

The owner can call the function `withdraw()` multiple times for the same IDO. If there are placed IDOs with the same `tokenAddress`, this may lead to token withdrawal of other projects as well as it gives the possibility to drain all `rasingToken` from the contract even the tokens of projects with an unended sale. If so, the contract balance may be insufficient to refund participants of removed IDOs.

```
function withdraw(uint256 _ido,address payable _recipient) public onlyOwner {
    Projects storage project = projects[_ido];
    require(project.supply > 0, "Project not found");
    require(block.timestamp > project.dates.saleEnd);
    IERC20 token = IERC20(project.tokenAddress);
    token.safeTransfer(_msgSender(), project.supply - project.soldAmount);
    rasingToken.safeTransfer(_recipient, (project.soldAmount.mul(project.price)) / 10 **
6);
}
```

Recommendation

In struct `Projects` add a flag that tells if this IDO was withdrawn or not.

C1-04 Staking rewards are not guaranteed ● High ✓ Acknowledged

The balance of the contract may be insufficient to pay the staking rewards, and users risk losing their funds as the source of income is uncertain and funds of stakers will be engaged in reward payments.

```
function unstakeToken() public {
    Stake storage stake = stakes[_msgSender()];
    require(stake.amount > 0, "No active stake!");
    ...
    if((block.timestamp - stake.time) / 60 / 60 / 24 >= LockPeriod){
        uint256 pft = (getPercent(stake.amount, APY) * ((block.timestamp - stake.time) /
60 / 60 / 24)) / 365;
        uint256 _amount = stake.amount;
        stake.amount = 0;
        stakingToken.safeTransfer(_msgSender(), _amount + pft);
    }else{
        uint256 _amount = stake.amount;
        stake.amount = 0;
        stakingToken.safeTransfer(_msgSender(), getPercent(_amount, 97));
    }
}
```

Recommendation

Due to rewards source ambiguity, we recommend holding rewards on a special fund wallet and transferring beneficial payment parts from it. It will ensure staked funds' safety and keep build-in logic.

C1-05 Getting back tokens when removing IDO ● High ✓ Resolved

In the function `removeIDO()` there is no transfer of the IDO token. Allocated for project tokens will be locked on the contract.

```
function removeIDO(uint256 _project) public onlyOwner{
    Projects storage project = projects[_project];
    require(block.timestamp < project.dates.saleEnd, "Can't delete IDO after it has
ended!");

    if(block.timestamp > project.dates.saleStart){
        for(uint i = 0; i < project.investors.length; i++){
            rasingToken.safeTransfer(project.investors[i],
project.users[project.investors[i]].usedAllocation);
        }
    }
    delete projects[_project];
}
```

Recommendation

Add transfer of IDO tokens inside this function.

C1-06 DoS with block gas limit function removeIDO() ● High ✓ Resolved

If there is a significant number of investors for an IDO and there is a need to cancel this IDO, there may be a possibility that the `removeIDO()` function would exceed the block gas limit while iterating over the list of investors. Therefore if a project will lose the removal option when the number of depositors hits the critical number.

```
function removeIDO(uint256 _project) public onlyOwner{
    Projects storage project = projects[_project];
    require(block.timestamp < project.dates.saleEnd, "Can't delete IDO after it has
ended!");

    if(block.timestamp > project.dates.saleStart){
        for(uint i = 0; i < project.investors.length; i++){
            rasingToken.safeTransfer(project.investors[i],
project.users[project.investors[i]].usedAllocation);
        }
    }
    delete projects[_project];
}
```

More information about the issue can be found at SwcRegistry.

Recommendation

In the struct `Projects` add a field that tells if an IDO was removed or not. Also, split the refund functionality to a separate function where users could return placed in the removed IDO assets.

C1-07 Access to change weights

● High ✓ Acknowledged

The owner can break the math of weights in staking in function `unstakeToken()`.

```
//let owner to change rank weights
function setWeights(uint256 _weight1, uint256 _weight2, uint256 _weight3, uint256
_weight4) public onlyOwner{
    weight1 = _weight1;
    weight2 = _weight2;
    weight3 = _weight3;
    weight4 = _weight4;
}
```

This will lead to funds loss of stakers. For example:

Rank 1 - 50 tokensRank 2 - 100 tokensWeight 1 - 10Weight 2 - 30

The user stakes 100 tokens and gets 30 weights (total weight is 30 too). Then the owner changes weights for example to this:

Weight 1 - 30Weight 2 - 50

After that, the user can't unstake his tokens because on line 343 transaction will fail. The contract will try to subtract 50 from 30.

Also changing global variables `FCFSMultiplier` and `allocMultiplier` also can break the math.

The same in function `remainingAlloc()` on line 253.

Recommendation

Delete the functions `setRanks()`, `setWeights()`, and `setMultiplier()`. In reality, they can't be changed safely because the contract must have zero stakes and zero IDOs to make it safely.

Update

After the update `totalWeight` global variable is changed but in all calculations `totalWeight` variable from struct `Projects` is used instead and because of that, this issue is still open.

C1-08 Elements are not popped from the array ● Medium ✓ Resolved

The function `remKYC()` implements deleting elements from the `KYC_Verified` array incorrectly. The array will have zero elements inside it. This will lead to gas consumption inflation. The same for the function `remWhite()`.

Recommendation

Implement the removing array element. The best practice is shown in the 'remove array element by copying the last element into to the place' section here.

C1-09 DoS with block gas limit ● Medium Partially fixed

Possibility of DoS with block gas limit (see SwcRegistry). These functions can consume a lot of gas or even be inaccessible:

1. `participate()`
2. `addWhite()`
3. `remWhite()`
4. `addKYC()`
5. `remKYC()`
6. `addIDO()`
7. `isKYC_Verified()`
8. `isWhiteListed()`
9. `isLotteryMember()`
10. `addLottery()`
11. `remLottery()`
12. `removeIDO()`
13. `stakeToken()`

14. unstakeToken()
15. adminRefund()

Recommendation

Rewrite logic of whitelist, lottery, and KYC using OpenZeppelin library EnumerableSet. This will allow removing for loops in functions isKYC_Verified(), isWhiteListed(), isLotteryMember(), remLottery(), remWhite(), and remKYC().

Update

Array iteration that could cause DoS was removed from the functions listed in the issue, however now it presences in adminRefund() function added in the update.

C1-10 Timestamp may be out of time intervals ● Medium Acknowledged

The case when block.timestamp is equal to project.dates.fcfsPrepare is not handled in participate() function. Transactions with such timestamps will affect neither user's usedFCFS nor project's FCFS.

```
function participate(uint256 _ido,uint256 _USD_Value) public returns(bool){
    Projects storage project = projects[_ido];
    ...
    project.users[_msgSender()].usedAllocation += _USD_Value;

    if(block.timestamp > project.dates.fcfsPrepare){
        project.users[_msgSender()].usedFCFS += _USD_Value;
    }

    project.soldAmount = project.soldAmount.add(TKN_Value);

    if(block.timestamp < project.dates.fcfsPrepare){
        project.FCFS = ((project.supply - project.soldAmount).mul(project.price)) / 10 **
6;
    }

    rasingToken.safeTransferFrom(_msgSender(), address(this), _USD_Value);
    return true;
```

```
  }
```

Recommendation

Correctly process the timestamp by adding it to one of the intervals simply making the inequality sign unstrict.

Update

Within the update were modified time intervals inside the requirements, but if statement still misses appropriate inequality sign. Change `<` on `<=` on L152.

C1-11 Default variables visibility ● Low Partially fixed

A lot of global variables have default visibility which is `internal`. Regard, users won't be able to see these variables in the blockchain explorer.

C1-12 No events ● Low Partially fixed

We recommend emitting events on important value changes to be easily tracked off-chain. No events are emitted in important contract functions.

C1-13 Unnecessary SafeMath ● Low ✓ Resolved

Starting from 0.8.0 solidity there is no need for `SafeMath` library, it is embedded into the compiler.

C1-14 No required function ● Low Acknowledged

a. There are no functions that return the length of `KYC_Verified`, `whiteListed`, and `lotteryMembers` arrays;

b. There is no function that tells how many tokens a user can claim from an IDO.

C1-15 Lack of validation in addIDO()

● Low ✓ Resolved

There is no check that the variable `_vestnig[2]` is not zero. With a zero variable, the function will loop and exceed the block gas limit. Transaction will be reverted without the real reason in the error message;

C1-16 Gas optimization

● Low ✓ Partially fixed

a. Variables `APY` and `LockPeriod` can be constant;

b. Consider using mapping or EnumerableSet from OpenZeppelin for checking if some user is KYC verified or not in functions `isKYC_Verified()`, `isWhiteListed()`, and `isLotteryMember()`, `remLottery()`. Using an array is much more expensive;

c. Lines 154-157 can be done without while loop;

d. Multiple storage reads:

- In the function `participate()` global variables `project.users[_msgSender()].usedAllocation`, `project.dates.fcfsPrepare`, `project.soldAmount`, `project.supply`, and `project.price`;
- In the function `participate()` global variables `project.users[_msgSender()].usedAllocation`, `project.dates.fcfsPrepare`, `project.soldAmount`, `project.supply`, and `project.price`;
- In the function `claimTokens()` global variables `project.dates.saleEnd`, `project.users[_msgSender()].claimed`, `project.vesting.vpDays`, and `project.vesting.vest2Precent`;
- In the function `withdraw()` global variables `project.supply` and `project.soldAmount`;
- In the function `isKYC_Verified()` global variable `KYC_Verified.length` is read multiple times. The same in functions `isWhiteListed()`, `remLottery()` and `isLotteryMember()`;
- In the function `stakeToken()` global variables stake.amount and `totalWeight`. Also, they are read after write;
- In the function `unstakeToken()` global variables `stake.amount` and `stake.time`;
- In the function `removeIDO()` global variables `project.investors.length` and `rasingToken`

;

- In the function `addWhite()` global variable `whiteListCount` is read after writing;
- In the function `remWhite()` global variables `whiteListed.length` and `whiteListCount` ;
- In the modifier `onlyOwner()` global variable `owner`.

C1-17 No revert messages ● Low ✓ Acknowledged

In some `require` statements in functions `claimTokens()`, `userRefund()`, `adminRefund()`, and `addIDO()` there are no revert messages.

C1-18 Custom ownable functionality ● Info ✓ Acknowledged

Better to use the OpenZeppelin library for ownable functionality. The implemented one has an unnecessary check on `address(1)` during ownership transfer and allows `msg.sender` to be zero address in `onlyOwner()` modifier.

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0) && newOwner != address(1), "Ownable: new owner is the
zero address");
    owner = newOwner;
}

modifier onlyOwner() {
    require(owner == _msgSender() || owner == address(0), "Ownable: caller is not the
owner");
    _;
}
```

C1-19 No support of tokens with commissions ● Info ✓ Acknowledged

The launchpad doesn’t support tokens with commissions, rebase, or RFI.

5. Conclusion

7 high, 3 medium, 7 low severity issues were found during the audit. 5 high, 1 medium, 2 low issues were resolved in the update.

Users willing to stake on the platform should consider that the staking tokenomics model has a serious design problem that may cause some users' funds loss.

This audit includes recommendations on code improvement and the prevention of potential attacks. We recommend adding tests with coverage of at least 90% with any updates in the future.

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