BLOCKHAT
SECURITY

YouToken

Smart Contract Security Audit

Prepared by BlockHat

July 4th, 2023 - July 7th, 2023

BlockHat.io

contact@blockhat.io

Document Properties

Client	YouWho
Version	1.0
Classification	Public

Scope

The YouToken Contract in the YouToken Repository

Link	Address
https://www.bscscan.com/token/0xB583961E033Dfe0FfF161952f7BA21c411b6103d#code	0xB583961E033Dfe0FfF161952f7BA21c411b6103d

Files	MD5 Hash
/YouToken.sol	ba5069a389a42bfec2386160e451f08f

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1 Introduction

YouToken engaged BlockHat to conduct a security assessment on the YouToken beginning on July 4th, 2023 and ending July 7th, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About YouToken

youwho is the world's first Proof of Participation based Decentralised Finance Ecosystem aiming to tackle Real World problems, we call this ReDeFi. While most DeFi platforms are reiterations of existing DeFi applications such as: yield farming, staking, token exchanges, NFT marketplaces, etc., we are looking to break the mould by utilising this amazing technology that is cryptocurrencies and decentralised finance to solve real world problems such as unemployment and banking the unbanked, which we believe go hand in hand. We aim to empower anyone to be their own Boss by combining the concepts behind the world's leading marketplaces, social networks, and banking platforms such as Airbnb, Uber, Grab, Ebay, Amazon, Fiverr, Facebook, and Paypal under one ecosystem, while utilising cryptocurrencies to settle payments worldwide

Issuer	YouWho
Website	`https://why.youwho.io/`
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

BlockHat used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

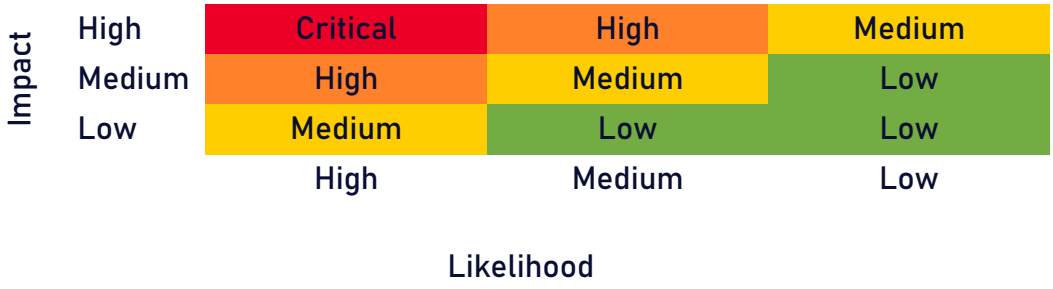
1.2.1 Risk Methodology

Vulnerabilities or bugs identified by BlockHat are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.



2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the YouToken implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 1 high-severity, 1 medium-severity, 1 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Centralization risk	HIGH	Not fixed
Cap Change not Controlled	MEDIUM	Not fixed
Floating Pragma	LOW	Not fixed

3 Finding Details

A YouToken.sol

A.1 Centralization risk [HIGH]

Description:

In the current implementation of the smart contract, a single address (_msgSender() which is the deployer of the contract) is assigned all roles (DEFAULT_ADMIN_ROLE, MINTER_ROLE, PAUSER_ROLE, CAP_ROLE). This implies a centralization of power.

Code:

Listing 1: YouToken.sol

```
1641     constructor() ERC20("youwho", "YOU") {
1642         _cap = 51 * 10**9 * (10**uint256(decimals()));
1643         _mint(_msgSender(), _cap * 70 / 100 );
1644         _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
1645         _setupRole(MINTER_ROLE, _msgSender());
1646         _setupRole(PAUSER_ROLE, _msgSender());
1647         _setupRole(CAP_ROLE, _msgSender());
1648     }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Consider assigning roles to different entities to distribute power and add more security to your smart contract. Single address control of all functions is generally not a good practice as it creates a single point of failure.

Status - Not fixed

A.2 Cap Change not Controlled [MEDIUM]

Description:

The changeCap function allows changing the maximum supply of tokens, but it doesn't prevent drastic changes. The cap could be changed to a very high or very low number, impacting tokenomics.

Code:

Listing 2: YouToken.sol

```
1650     function changeCap(uint256 newCap) external {
1651         require(hasRole(CAP_ROLE, _msgSender()),"Error Cap: You must have
                 ↪  cap role to change cap");
1652         require(ERC20.totalSupply() <= newCap,"Error Cap: New cap cant be
                 ↪  less than current total supply");
1653         _cap = newCap;
1654     }
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Given that you've stated the token supply cap in your white paper, it would be best to honor that promise to maintain trust with your users. Rather than allowing changes to the cap, you should consider setting a fixed cap that aligns with what was mentioned in your white paper. If there is a strong reason to change the cap, it should be done with full transparency, perhaps using a governance mechanism where token holders can vote on proposed changes.

Status - Not fixed

A.3 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensuring that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Code:

Listing 3: YouToken.sol

```
1609 pragma solidity ^0.8.0;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma not be used in production.Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Not fixed

4 Static Analysis (Slither)

Description:

Block Hat expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
INFO:Detectors:
AccessControlEnumerable._grantRole(bytes32,address) (YouToken.sol
   ↪ #922-925) ignores return value by _roleMembers[role].add(account)
   ↪  (YouToken.sol#924)
AccessControlEnumerable._revokeRole(bytes32,address) (YouToken.sol
   ↪ #930-933) ignores return value by _roleMembers[role].remove(
   ↪ account) (YouToken.sol#932)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #unused-return
INFO:Detectors:
EnumerableSet.values(EnumerableSet.AddressSet) (YouToken.sol#280-289)
   ↪ uses assembly
       - INLINE ASM (YouToken.sol#284-286)
EnumerableSet.values(EnumerableSet.UintSet) (YouToken.sol#353-362) uses
   ↪ assembly
       - INLINE ASM (YouToken.sol#357-359)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #assembly-usage
INFO:Detectors:
AccessControl._setRoleAdmin(bytes32,bytes32) (YouToken.sol#839-843) is
   ↪ never used and should be removed
```

```
Context._msgData() (YouToken.sol#640-642) is never used and should be
    ↪ removed
EnumerableSet._values(EnumerableSet.Set) (YouToken.sol#148-150) is never
    ↪  used and should be removed
EnumerableSet.add(EnumerableSet.Bytes32Set,bytes32) (YouToken.sol
    ↪ #164-166) is never used and should be removed
EnumerableSet.add(EnumerableSet.UintSet,uint256) (YouToken.sol#303-305)
    ↪ is never used and should be removed
EnumerableSet.at(EnumerableSet.Bytes32Set,uint256) (YouToken.sol
    ↪ #202-204) is never used and should be removed
EnumerableSet.at(EnumerableSet.UintSet,uint256) (YouToken.sol#341-343)
    ↪ is never used and should be removed
EnumerableSet.contains(EnumerableSet.AddressSet,address) (YouToken.sol
    ↪ #247-249) is never used and should be removed
EnumerableSet.contains(EnumerableSet.Bytes32Set,bytes32) (YouToken.sol
    ↪ #181-183) is never used and should be removed
EnumerableSet.contains(EnumerableSet.UintSet,uint256) (YouToken.sol
    ↪ #320-322) is never used and should be removed
EnumerableSet.length(EnumerableSet.Bytes32Set) (YouToken.sol#188-190) is
    ↪  never used and should be removed
EnumerableSet.length(EnumerableSet.UintSet) (YouToken.sol#327-329) is
    ↪ never used and should be removed
EnumerableSet.remove(EnumerableSet.Bytes32Set,bytes32) (YouToken.sol
    ↪ #174-176) is never used and should be removed
EnumerableSet.remove(EnumerableSet.UintSet,uint256) (YouToken.sol
    ↪ #313-315) is never used and should be removed
EnumerableSet.values(EnumerableSet.AddressSet) (YouToken.sol#280-289) is
    ↪  never used and should be removed
EnumerableSet.values(EnumerableSet.Bytes32Set) (YouToken.sol#214-216) is
    ↪  never used and should be removed
EnumerableSet.values(EnumerableSet.UintSet) (YouToken.sol#353-362) is
    ↪ never used and should be removed
Strings.toHexString(uint256) (YouToken.sol#465-476) is never used and
    ↪ should be removed
```

```
Strings.toString(uint256) (YouToken.sol#440-460) is never used and
    ↪ should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code
INFO:Detectors:
Pragma version^0.8.0 (YouToken.sol#10) allows old versions
Pragma version^0.8.0 (YouToken.sol#370) allows old versions
Pragma version^0.8.0 (YouToken.sol#398) allows old versions
Pragma version^0.8.0 (YouToken.sol#429) allows old versions
Pragma version^0.8.0 (YouToken.sol#499) allows old versions
Pragma version^0.8.0 (YouToken.sol#590) allows old versions
Pragma version^0.8.0 (YouToken.sol#623) allows old versions
Pragma version^0.8.0 (YouToken.sol#650) allows old versions
Pragma version^0.8.0 (YouToken.sol#875) allows old versions
Pragma version^0.8.0 (YouToken.sol#941) allows old versions
Pragma version^0.8.0 (YouToken.sol#1034) allows old versions
Pragma version^0.8.0 (YouToken.sol#1119) allows old versions
Pragma version^0.8.0 (YouToken.sol#1149) allows old versions
Pragma version^0.8.0 (YouToken.sol#1534) allows old versions
Pragma version^0.8.0 (YouToken.sol#1569) allows old versions
Pragma version^0.8.0 (YouToken.sol#1609) allows old versions
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity
INFO:Detectors:
Variable YouToken._cap (YouToken.sol#1639) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #conformance-to-solidity-naming-conventions
INFO:Slither:YouToken.sol analyzed (16 contracts with 85 detectors), 41
    ↪ result(s) found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

5 Conclusion

We examined the design and implementation of YouToken in this audit and found several issues of various severities. We advise YouWho team to implement the recommendations contained in all 3 of our findings to further enhance the code's security. It is of utmost priority to start by addressing the most severe exploit discovered by the auditors then followed by the remaining exploits, and finally we will be conducting a re-audit following the implementation of the remediation plan contained in this report.

We would much appreciate any constructive feedback or suggestions regarding our methodology, audit findings, or potential scope gaps in this report.

BLOCKHAT
SECURITY

For a Contract Audit, contact us at contact@blockhat.io