QuillAudits

Audit Report

September, 2020

YFIW Finance

Contents

Introduction

This Audit Report highlights the overall security of YFIW Smart Contract. With this report, we have tried to ensure the reliability of their smart contract by complete assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Quillhash team has performed thorough testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the process.
- Analysing the complexity of the code by thorough, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests
- Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Audit Details

Project Name: YFIW
Website/Etherscan Code (Mainnet): 0x065ccf8d682995be66e38eaee3c2475f636634ab
Website/Etherscan Code (Kovan): 0x3580207BD5a5368e3021896808f719F0734AC35c
Languages: Solidity (Smart contract), Javascript (Unit Testing)
Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Securify, Mythril, Contract Library, Slither

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security
Identifying security related issues within each contract and the system of contracts.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Correctness
- Sections of code with high complexity
- Readability
- Quantity and quality of test coverage

Security

Every issue in this report was assigned a severity level from the following:

High level severity issues
Issues on this level are critical to the smart contract’s performance/ functionality and should be fixed before moving to a live environment.

Medium level severity issues
They could potentially bring problems and should eventually be fixed.

Low level severity issues
They are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

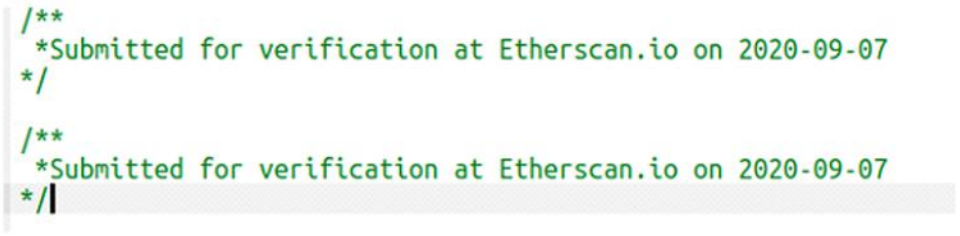
	Low	Medium	High
Open	7	5	2
Closed	0	0	0

Manual Audit

For this section the code was tested/read line by line by our developers. We also used Remix IDE's JavaScript VM and Kovan networks to test the contract functionality.

Low Level Severity Issues

1. Duplicate comments about Etherscan verification date have been specified. It is recommended to remove one of them.



```
/**
 *Submitted for verification at Etherscan.io on 2020-09-07
*/

/**
 *Submitted for verification at Etherscan.io on 2020-09-07
*/
```

2. We recommend using OpenZeppelin's IERC20 interface instead of "abstract contract ERC20". It offers numerous benefits over the current implementation:

    2.1 Since all the functions defined within the contract are abstract it should be defined as an interface.
    2.2 All the functions within an interface are virtual by default and there will be no need to specify it explicitly.
    2.3 All the functions would have external visibility and will consume less gas.
    2.4 The variable totalSupply will be replaced with an external function which will increase privacy and save gas during execution.

   We also recommend using OpenZeppelin's EC20.sol instead of Token contract. Instead of providing benefits similar to the ones specified above it also provides increaseAllowance and decreaseAllowance which makes the overall "approve" nomenclature lucid. The inherited owned contract within token contract can also be inherited within YFIW contract directly.

3. YFIW is a confusing name, the contracts name should not be an acronym it should be specified in full using camel case (camelCase) pattern.

4. All the “require” statements used in the contract should also specify error messages for easy debugging.

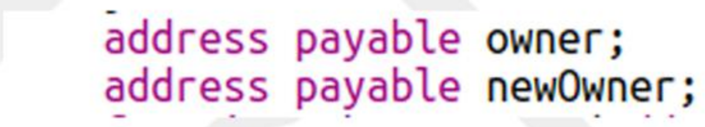
5. Natspecs should be used to improve code readability.

6. The pragama versions used within these contracts are too recent and are not locked as well. Consider using version 0.5.11 for deploying the contracts. This change will affect contract code as well..

7. The current owner of the contract can change the ownership of the contract to a new address (let's call him x). For this the current owner will need to call the changeOwner function of the contract. But the actual ownership won't be transferred until the new owner (x) calls acceptOwnership function. Between this delay the previous owner will keep receiving new ethers transferred to contract address.

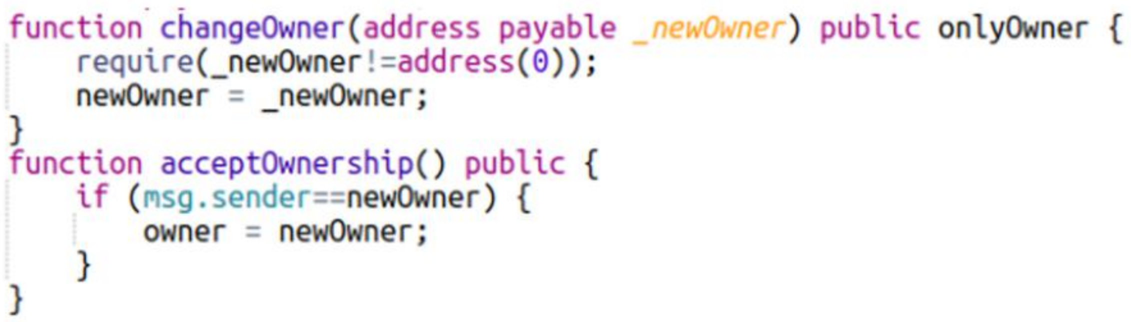
Medium Level Severity Issues

1. Visibility of variables should be explicitly specified. The variables owner and newOwner have been defaulted to internal and therefore cannot be read via contract calls.



```
address payable owner;
address payable newOwner;
```

2. Visibility of changeOwner as well as acceptOwnership function should be updated to “external” from “public”. As these functions have not been called by any other contract function. This would help save gas during function calling.



```
function changeOwner(address payable _newOwner) public onlyOwner {
    require(_newOwner!=address(0));
    newOwner = _newOwner;
}
function acceptOwnership() public {
    if (msg.sender==newOwner) {
        owner = newOwner;
    }
}
```

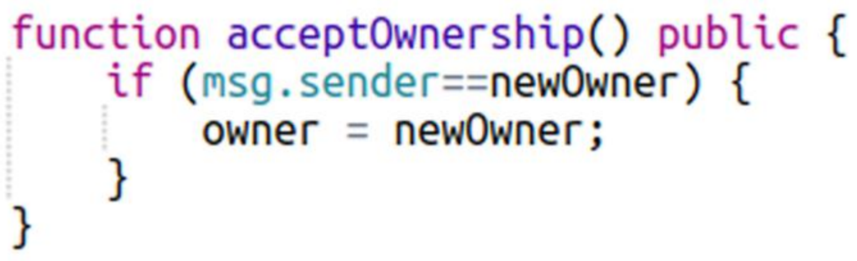
This is important because the owner is assigned the initial token supply, and can only transfer ownership to a new address. Therefore the address information is recommended to be made public via the contract.

The following list of functions should be made external for saving gas during function calls:

- changeOwner(address)
- acceptOwnership()
- balanceOf(address)
- transfer(address,uint256)
- transferFrom(address,address,uint256)
- approve(address,uint256)
- allowance(address,address)

3. YFIW contract does not use custom events specific to the contract functionality. Events should be fired with all state variable updates as good practise. This makes it easier to build dApps on top of the contract’s using existing tools. For instance when the owner is changed or when the contract receives ethers.

4. Function acceptOwnership should revert if the message sender is not the newOwner. Instead of wasting 22311 gas as transaction cost. There is no simple way to identify if the function execution was successful or a failure.

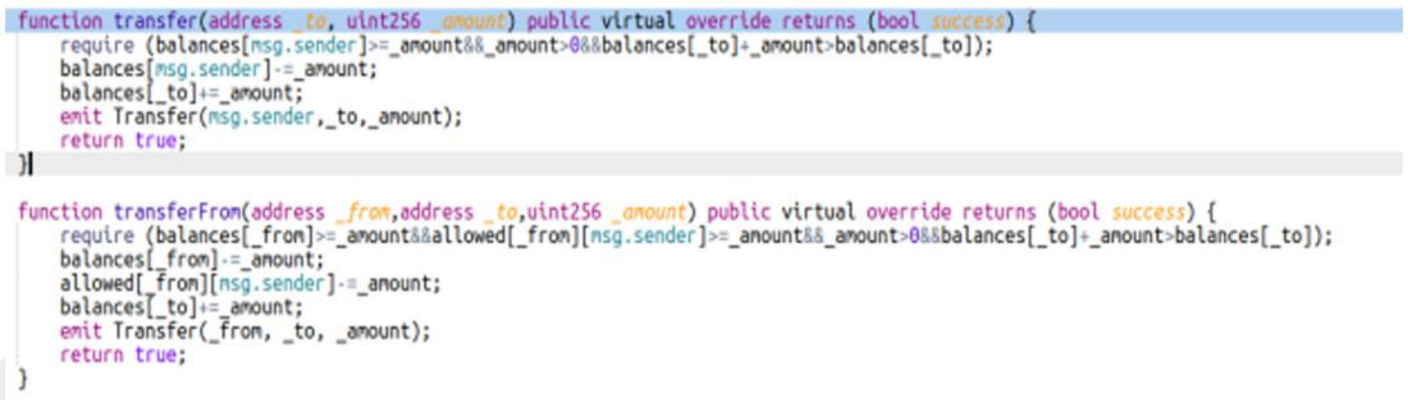


```
function acceptOwnership() public {
    if (msg.sender==newOwner) {
        owner = newOwner;
    }
}
```

5. A check should be placed to revert if the input argument to a function is a zeroAddress. The check should be applied to wherever addresses are taken as input, which includes: transfer, transferFrom and approve.

High Level Severity Issues

1. Should use OpenZeppelin's SafeMath.sol to avoid possible integer underflow/overflow.



```
function transfer(address _to, uint256 _amount) public virtual override returns (bool success) {
    require (balances[msg.sender]>=_amount&&_amount>0&&balances[_to]+_amount>balances[_to]);
    balances[msg.sender]-=_amount;
    balances[_to]+=_amount;
    emit Transfer(msg.sender,_to,_amount);
    return true;
}

function transferFrom(address _from,address _to,uint256 _amount) public virtual override returns (bool success) {
    require (balances[_from]>=_amount&&allowed[_from][msg.sender]>=_amount&&_amount>0&&balances[_to]+_amount>balances[_to]);
    balances[_from]-=_amount;
    allowed[_from][msg.sender]-=_amount;
    balances[_to]+=_amount;
    emit Transfer(_from, _to, _amount);
    return true;
}
```

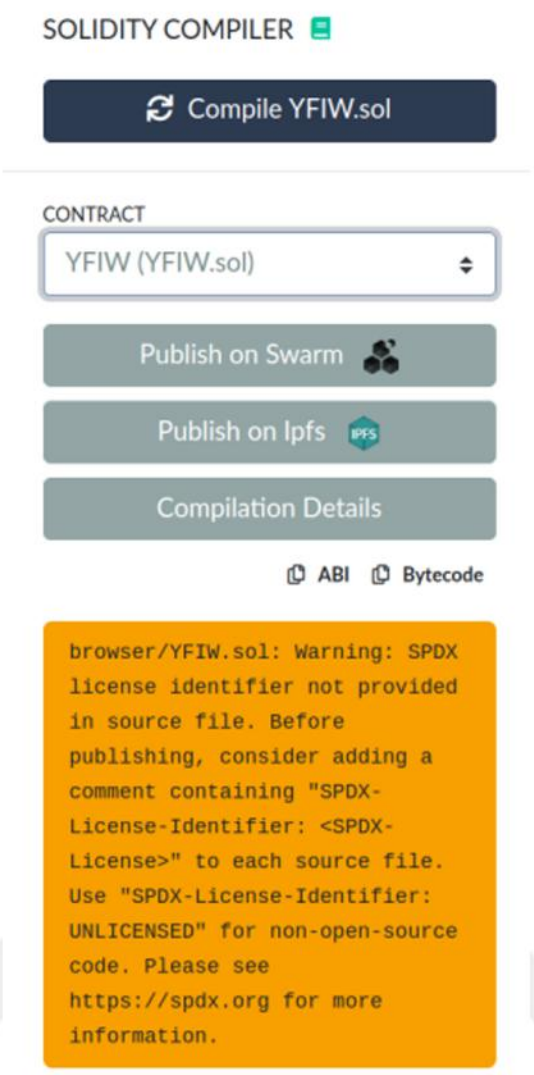
The underflow/overflow issue could cause a catastrophic loss for the token holder's account, and make a hacker rich effortlessly. An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.

2. Function transferFrom should return an Approve event showcasing new Transfer value. During transferFrom we use the approved allowance of a Token holder and spend it, this new reduced value should be emitted via an event in transferFrom.

Automated Audit

Remix Compiler Warnings

It throws warnings by Solidity's compiler. If it encounters any errors the contract cannot be compiled and deployed.



```
browser/YFIW.sol: Warning: SPDX
license identifier not provided
in source file. Before
publishing, consider adding a
comment containing "SPDX-
License-Identifier: <SPDX-
License>" to each source file.
Use "SPDX-License-Identifier:
UNLICENSED" for non-open-source
code. Please see
https://spdx.org for more
information.
```

It is recommended to add the following line at the beginning of contract code:
// SPDX-License-Identifier: MIT

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Multiple Linting violations were detected by Solhint, the linted version of the contract is as follows:

```
/**
*Submitted for verification at Etherscan.io on 2020-09-07
*/

/**
*Submitted for verification at Etherscan.io on 2020-09-07
*/

pragma solidity ^0.6.7;

contract Owned {
  modifier onlyOwner() {
      require(msg.sender == owner);
      _;
  }
  address payable owner;
  address payable newOwner;

  function changeOwner(address payable _newOwner) public onlyOwner {
      require(_newOwner != address(0));
      newOwner = _newOwner;
  }

  function acceptOwnership() public {
      if (msg.sender == newOwner) {
          owner = newOwner;
      }
  }
}

abstract contract ERC20 {
  uint256 public totalSupply;

  function balanceOf(address _owner)
      public
      virtual
      view
      returns (uint256 balance);

  function transfer(address _to, uint256 _value)
      public
      virtual
      returns (bool success);

  function transferFrom(
      address _from,
      address _to,
      uint256 _value
  ) public virtual returns (bool success);
  function approve(address _spender, uint256 _value)
    public
    virtual
    returns (bool success);

  function allowance(address _owner, address _spender)
    public
    virtual
    view
    returns (uint256 remaining);

  event Transfer(address indexed _from, address indexed _to, uint256 _value);
  event Approval(
    address indexed _owner,
    address indexed _spender,
    uint256 _value
  );
}

contract Token is Owned, ERC20 {
  string public symbol;
  string public name;
  uint8 public decimals;
  mapping(address => uint256) balances;
  mapping(address => mapping(address => uint256)) allowed;

  function balanceOf(address _owner)
    public
    virtual
    override
    view
    returns (uint256 balance)
  {
    return balances[_owner];
  }

  function transfer(address _to, uint256 _amount)
    public
    virtual
    override
    returns (bool success)
  {
    require(
      balances[msg.sender] >= _amount &&
        _amount > 0 &&
        balances[_to] + _amount > balances[_to]
    );
    balances[msg.sender] -= _amount;
    balances[_to] += _amount;
    emit Transfer(msg.sender, _to, _amount);
    return true;
  }

  function transferFrom(
    address _from,
    address _to,
    uint256 _amount
  ) public virtual override returns (bool success) {
    require(
      balances[_from] >= _amount &&
        allowed[_from][msg.sender] >= _amount &&
        _amount > 0 &&
            balances[_to] + _amount > balances[_to]
        );
        balances[_from] -= _amount;
        allowed[_from][msg.sender] -= _amount;
        balances[_to] += _amount;
        emit Transfer(_from, _to, _amount);
        return true;
    }

    function approve(address _spender, uint256 _amount)
        public
        virtual
        override
        returns (bool success)
    {
        allowed[msg.sender][_spender] = _amount;
        emit Approval(msg.sender, _spender, _amount);
        return true;
    }

    function allowance(address _owner, address _spender)
        public
        virtual
        override
        view
        returns (uint256 remaining)
    {
        return allowed[_owner][_spender];
    }
}

contract YFIW is Token {
    constructor() public {
        symbol = "YFIW";
        name = "YFIW.FINANCE";
        decimals = 18;
        totalSupply = 29330000000000000000000;
        owner = msg.sender;
        balances[owner] = totalSupply;
    }

    receive() external payable {
        require(msg.value > 0);
        owner.transfer(msg.value);
    }
}
```

Securify

Securify is a tool that scans Ethereum smart contracts for critical security vulnerabilities. Securify statically analyzes the EVM code of the smart contract to infer important semantic information (including control-flow and data-flow facts) about the contract. The report provided by Securify can be accessed here:
https://securify.chainsecurity.com/report/c7bb458a58a2ef3b3662dfcf416d471 c3c9ef2381ad1c70c0d71b3e68b2fe946

Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to these in real time.

We performed analysis using contract Library on the mainnet address of the YFIW contract: 0x065ccf8d682995be66e38eaee3c2475f636634ab

Analysis summary can be accessed here:
https://contract-library.com/contracts/Ethereum/0x065ccf8d682995be66e38eaee3c2475f636634ab

It did not return any issue during the analysis.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has the critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

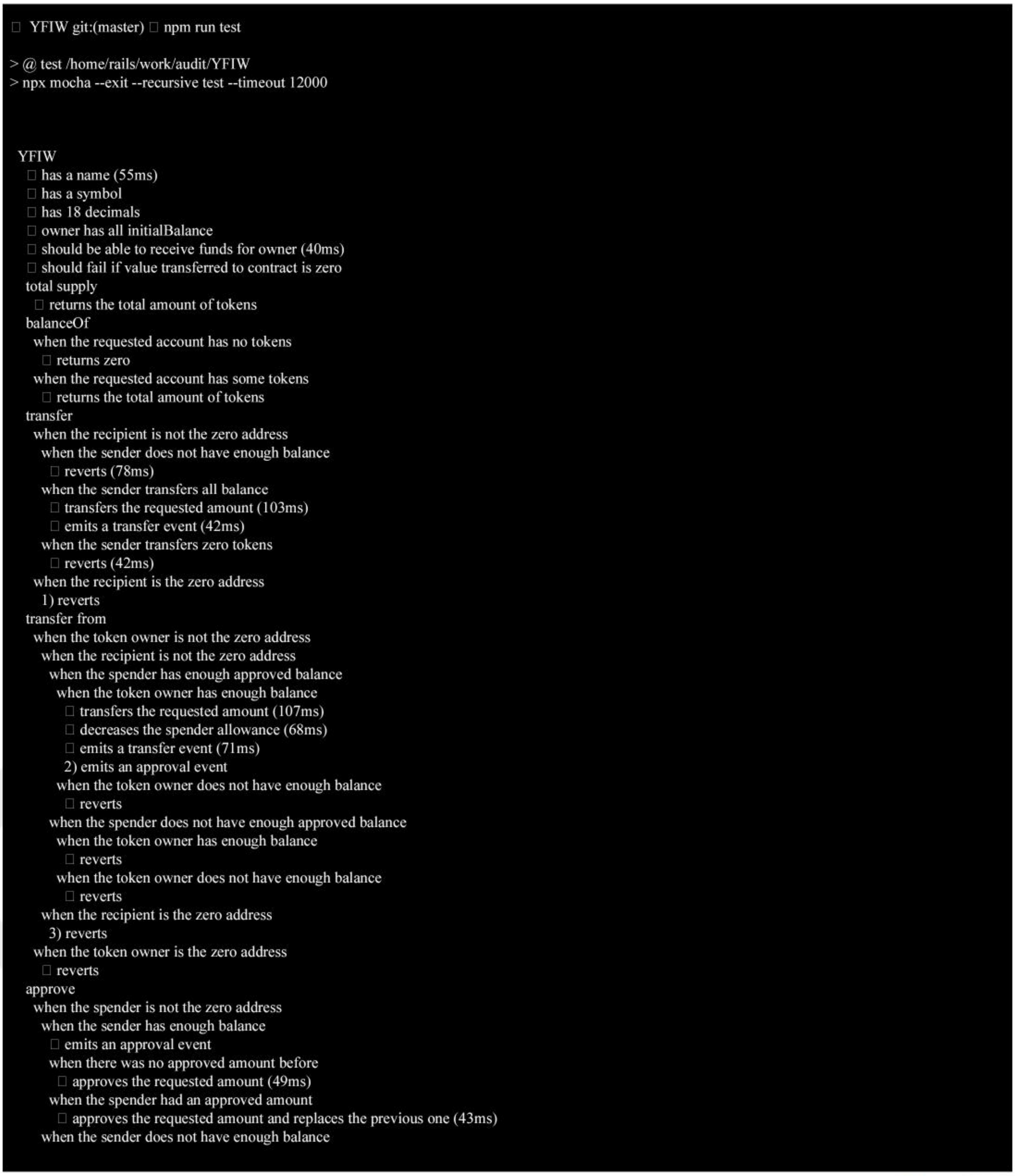
```
□  YFIW git:(master) □ slither .
'npx truffle compile --all' running (use --truffle-version truffle@x.x.x to use specific version)

Compiling your contracts...
===========================
> Compiling ./contracts/YFIW.sol
> Artifacts written to /home/rails/work/audit/YFIW/build/contracts
> Compiled successfully using:
   - solc: 0.6.7+commit.b8d736ae.Emscripten.clang


INFO:Detectors:
Pragma version^0.6.7 (YFIW.sol#9) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter Owned.changeOwner(address)._newOwner (YFIW.sol#19) is not in mixedCase
Parameter Token.balanceOf(address)._owner (YFIW.sol#77) is not in mixedCase
Parameter Token.transfer(address,uint256)._to (YFIW.sol#87) is not in mixedCase
Parameter Token.transfer(address,uint256)._amount (YFIW.sol#87) is not in mixedCase
Parameter Token.transferFrom(address,address,uint256)._from (YFIW.sol#105) is not in mixedCase
Parameter Token.transferFrom(address,address,uint256)._to (YFIW.sol#106) is not in mixedCase
Parameter Token.transferFrom(address,address,uint256)._amount (YFIW.sol#107) is not in mixedCase
Parameter Token.approve(address,uint256)._spender (YFIW.sol#122) is not in mixedCase
Parameter Token.approve(address,uint256)._amount (YFIW.sol#122) is not in mixedCase
Parameter Token.allowance(address,address)._owner (YFIW.sol#133) is not in mixedCase
Parameter Token.allowance(address,address)._spender (YFIW.sol#133) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
YFIW.constructor() (YFIW.sol#145-152) uses literals with too many digits:
    - totalSupply = 29330000000000000000000 (YFIW.sol#149)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
changeOwner(address) should be declared external:
    - Owned.changeOwner(address) (YFIW.sol#19-22)
acceptOwnership() should be declared external:
    - Owned.acceptOwnership() (YFIW.sol#24-28)
balanceOf(address) should be declared external:
    - Token.balanceOf(address) (YFIW.sol#77-85)
    - ERC20.balanceOf(address) (YFIW.sol#34-38)
transfer(address,uint256) should be declared external:
    - Token.transfer(address,uint256) (YFIW.sol#87-102)
    - ERC20.transfer(address,uint256) (YFIW.sol#40-43)
transferFrom(address,address,uint256) should be declared external:
    - Token.transferFrom(address,address,uint256) (YFIW.sol#104-120)
    - ERC20.transferFrom(address,address,uint256) (YFIW.sol#45-49)
approve(address,uint256) should be declared external:
    - ERC20.approve(address,uint256) (YFIW.sol#51-54)
    - Token.approve(address,uint256) (YFIW.sol#122-131)
allowance(address,address) should be declared external:
    - ERC20.allowance(address,address) (YFIW.sol#56-60)
    - Token.allowance(address,address) (YFIW.sol#133-141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-as-external
INFO:Slither:. analyzed (4 contracts with 46 detectors), 20 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Unit Tests

We used open zeppelin's test-helpers and test-environment to write these contract test cases and executed them using a combination of truffle, mocha and chai.

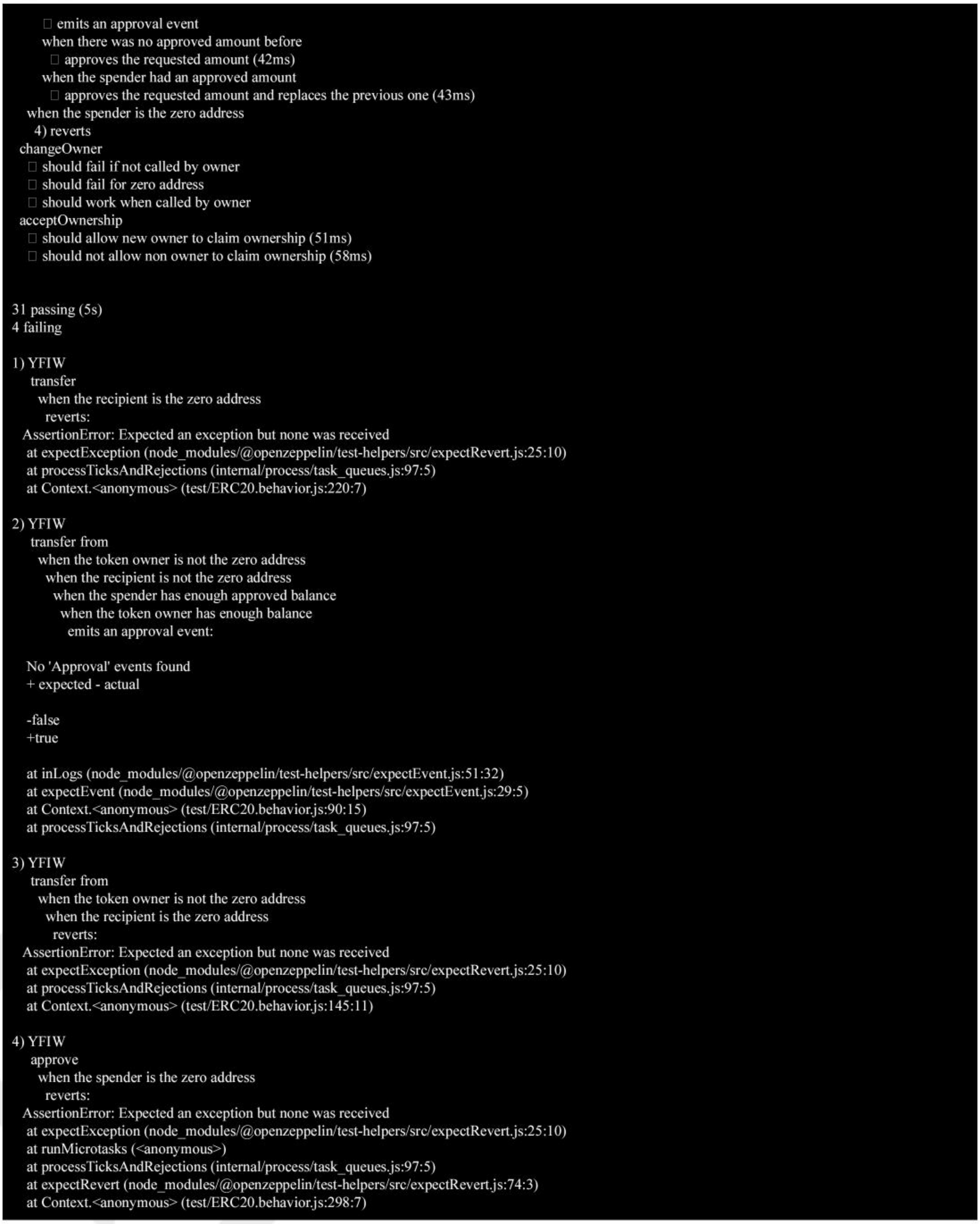


```
□ YFIW git:(master) □ npm run test

> @ test /home/rails/work/audit/YFIW
> npx mocha --exit --recursive test --timeout 12000



  YFIW
    □ has a name (55ms)
    □ has a symbol
    □ has 18 decimals
    □ owner has all initialBalance
    □ should be able to receive funds for owner (40ms)
    □ should fail if value transferred to contract is zero
    total supply
      □ returns the total amount of tokens
    balanceOf
      when the requested account has no tokens
        □ returns zero
      when the requested account has some tokens
        □ returns the total amount of tokens
    transfer
      when the recipient is not the zero address
        when the sender does not have enough balance
          □ reverts (78ms)
        when the sender transfers all balance
          □ transfers the requested amount (103ms)
          □ emits a transfer event (42ms)
        when the sender transfers zero tokens
          □ reverts (42ms)
      when the recipient is the zero address
        1) reverts
    transfer from
      when the token owner is not the zero address
        when the recipient is not the zero address
          when the spender has enough approved balance
            when the token owner has enough balance
              □ transfers the requested amount (107ms)
              □ decreases the spender allowance (68ms)
              □ emits a transfer event (71ms)
              2) emits an approval event
            when the token owner does not have enough balance
              □ reverts
          when the spender does not have enough approved balance
            when the token owner has enough balance
              □ reverts
            when the token owner does not have enough balance
              □ reverts
        when the recipient is the zero address
          3) reverts
      when the token owner is the zero address
        □ reverts
    approve
      when the spender is not the zero address
        when the sender has enough balance
          □ emits an approval event
          when there was no approved amount before
            □ approves the requested amount (49ms)
          when the spender had an approved amount
            □ approves the requested amount and replaces the previous one (43ms)
        when the sender does not have enough balance
```



```
      □ emits an approval event
      when there was no approved amount before
        □ approves the requested amount (42ms)
      when the spender had an approved amount
        □ approves the requested amount and replaces the previous one (43ms)
    when the spender is the zero address
      4) reverts
  changeOwner
    □ should fail if not called by owner
    □ should fail for zero address
    □ should work when called by owner
  acceptOwnership
    □ should allow new owner to claim ownership (51ms)
    □ should not allow non owner to claim ownership (58ms)


31 passing (5s)
4 failing

1) YFIW
     transfer
       when the recipient is the zero address
         reverts:
   AssertionError: Expected an exception but none was received
    at expectException (node_modules/@openzeppelin/test-helpers/src/expectRevert.js:25:10)
    at processTicksAndRejections (internal/process/task_queues.js:97:5)
    at Context.<anonymous> (test/ERC20.behavior.js:220:7)

2) YFIW
     transfer from
       when the token owner is not the zero address
         when the recipient is not the zero address
           when the spender has enough approved balance
             when the token owner has enough balance
               emits an approval event:

    No 'Approval' events found
    + expected - actual

    -false
    +true

    at inLogs (node_modules/@openzeppelin/test-helpers/src/expectEvent.js:51:32)
    at expectEvent (node_modules/@openzeppelin/test-helpers/src/expectEvent.js:29:5)
    at Context.<anonymous> (test/ERC20.behavior.js:90:15)
    at processTicksAndRejections (internal/process/task_queues.js:97:5)

3) YFIW
     transfer from
       when the token owner is not the zero address
         when the recipient is the zero address
           reverts:
   AssertionError: Expected an exception but none was received
    at expectException (node_modules/@openzeppelin/test-helpers/src/expectRevert.js:25:10)
    at processTicksAndRejections (internal/process/task_queues.js:97:5)
    at Context.<anonymous> (test/ERC20.behavior.js:145:11)

4) YFIW
     approve
       when the spender is the zero address
         reverts:
   AssertionError: Expected an exception but none was received
    at expectException (node_modules/@openzeppelin/test-helpers/src/expectRevert.js:25:10)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:97:5)
    at expectRevert (node_modules/@openzeppelin/test-helpers/src/expectRevert.js:74:3)
    at Context.<anonymous> (test/ERC20.behavior.js:298:7)
```

The four failing test cases represent standard violations in ERC20 behaviour as stated by OpenZeppelin and have been already covered in the manual analysis part.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the YFIW contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Summary

Use case of the smart contract is simple and the code is relatively small. Altogether, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. But there are a number of issues/vulnerabilities to be tackled in various severity levels, it is recommended to fix them before implementing a live version.

QuillAudits

448-A EnKay Square, Opposite Cyber Hub,
Gurugram, Harayana, India - 122016

audits.quillhash.com

hello@quillhash.com