Hadouken

Smart Contract Security Assessment

September 15, 2022

VERACITY

Contents

Disclaimer

Veracity Security (“Veracity”) has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature.

The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by the Veracity team.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided ‘as is’, and Veracity is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Veracity or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Overview

This report has been prepared for Hadouken. Veracity provides an examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

Name	Hadouken
URL	https://hadoukeninu.io/
Platform	Ethereum
Language	Solidity

Classification of Issues

Severity	Description
● High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.
● Optimization	Suboptimal implementations that may result in additional gas consumption, unnecessary computation or avoidable inefficiencies.

Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change)
● High	5	5	0	0
● Medium	4	3	1	1
● Low	1	0	0	1
● Informational	2	0	0	2
Total	12	8	1	3

Hadouken

ID	Severity	Summary	Status
01	**HIGH**	Function transfer: Passing and returning _value to fee removal functions results in incorrect token allocations, sum balances and total supply divergence	**RESOLVED**
02	**HIGH**	ProcessSellBurn function does not burn any supply causing a mismatch between balances and totalSupply	**RESOLVED**
03	**HIGH**	Solidity compiler version used 0.7.0 does not include safemath for uint operations and SafeMath library not used.	**RESOLVED**
04	**HIGH**	Functions transfer, transferFrom: _value not updated when processing fees and reflections resulting in over allocation of tokens to receiving user.	**RESOLVED**
05	**MEDIUM**	Functions: All. Multiplication as a result of division will result in loss of precision and incorrect allocation of tokens.	**PARTIALLY RESOLVED**
06	**MEDIUM**	Incorrect ERC20 interface implemented causing contract interoperability risks	**RESOLVED**

07	MEDIUM	Function: transferFrom requires approval on msg.sender instead of _from causing interoperability problems with other contracts or wallets	RESOLVED
08	MEDIUM	Functions: Multiple - spurious transfer events will misreport activity to scanning and reporting systems.	RESOLVED
09	LOW	Function: transferFrom #156-205) performs a multiplication on the result of a division:	ACKNOWLEDGED
10	INFORMATIONAL	mixedCase variable and function names are used throughout.	ACKNOWLEDGED

Graph

ID	Severity	Summary	Status
11	INFORMATIONAL	mixedCase variable and function names are used throughout.	ACKNOWLEDGED
12	HIGH	Function: initialize is public without a guard meaning anyone can take ownership of the graph contract.	RESOLVED

Findings

The contracts assessed have been completely authored from scratch rather than using industry tested implementations for ERC20 or standard interfaces. This can result in the introduction of vulnerabilities or bugs that have not been seen or addressed in previous projects. However our team has made recommendations and several code sweeps to mitigate the effect of not using industry standard libraries.

Hadouken

This report has been prepared for Hadouken. Veracity provides an examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

Privileged Roles

The following functions can be called by the deployer or deployerALT of the Hadouken:

- SetDex
- configImmuneToMaxWallet
- renounceContract
- editThreshold
- SweepToken
- sweep

Initial Token Allocation

On deployment 1000000000 (10 Billion) tokens are minted to the contract deployer.

Fees, Taxes, Rules

ID	Rule	Value
01	**BuyFeePercent**	3
02	**SellFeePercent**	2
03	**ReflectBuyFeePercent**	3
04	**ReflectSellFeePercent**	2
05	**BuyLiqTax**	1
06	**SellLiqTax**	2
07	**maxWalletPercent**	2

01

HIGH SEVERITY

Function transfer: Passing and returning _value to fee removal functions results in incorrect token allocations, sum balances and total supply divergence

Description

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

```
_value = ProcessBuyFee(_value, msg.sender);
_value = ProcessBuyReflection(_value, msg.sender);
_value = ProcessBuyLiq(_value, msg.sender);
```

_value is recalculated in the called function

ProcessBuyFee decreases _value by 3%

so when it is passed to ProcessBuyReflection is 97% of the original value

ProcessBuyReflection decreased it by 3%, which is 3% of 97% = 2.91% not 3%. ProcessBuyLiq receives a value which is already decreased to 94.09% and it decreases the value to 93.14.

This results is incorrect number of allocated tokens and sum of balances diverging from total supply.

Recommendation

Use a temporary variable and return the fee from each call. Then subtract the total fee from _value.

Resolution

RESOLVED

They switched the calculation to use a temporary variable and deducted the full _value after the fees.

02

HIGH SEVERITY

ProcessSellBurn function does not burn any supply causing a mismatch between balances and totalSupply

Description

In the function ProcessSellBurn the fee is calculated and subtracted from the incoming _value, but the number of tokens in supply is not reduced.

Recommendation

Deduct the amount burned from the total supply.

Resolution

RESOLVED

totalSupply adjusted correctly

03

HIGH SEVERITY

Solidity compiler version used 0.7.0 does not include safemath for uint operations and SafeMath library not used.

Description

Throughout the contract no SafeMath library checks are used. There is a risk of both under and overflow errors in calculation with these checks.

Recommendation

Switch to compiler version >0.8.0 to use build in safe math operations for uint

Resolution

RESOLVED

Compiler compatibility switched to >0.8.0

04

HIGH SEVERITY

Functions transfer, transferFrom: _value not updated when processing fees and reflections resulting in over allocation of tokens to receiving user.

Description

154,209,220 do not update _value:

```
_value - feeamt;
```

This would result in over allocation to receiving user.

Recommendation

Fix _value adjustment to:

```
balances[_to] += _value;
```

Move accounting operations adjacent so it’s clear what is happening.

Resolution

RESOLVED

Accounting in transfer, transferFrom adjusted:

```
balances[msg.sender] -= _value;
_value -= feeamt;
balances[_to] += _value;
```

05

MEDIUM SEVERITY

Functions: All. Multiplication as a result of division will result in loss of precision and incorrect allocation of tokens.

Description

The multiplication as a result of division pattern is used throughout causing precision errors, potential incorrect allocation of fees/tokens.

Example in ProcessBuyReflection:

```
uint fee = ReflectBuyFeePercent*(_value/100);
```

Recommendation

Switch to multiplication first.

```
(ReflectSellFeePercent*_value)/100
```

Fix all instances.

Resolution

PARTIALLY RESOLVED

Fees and token allocations have been corrected, so precision is not assigned.

Some requires checks have not been updated, however this is not causing any accounting issues.

06

MEDIUM SEVERITY

Incorrect ERC20 interface implemented causing contract interoperability risks

Description

ERC20 interface and functions not returning the correct types on success/fail. This will cause interoperability problems with contracts and wallets.

```
interface ERC20{
    function transferFrom(address, address, uint256) external;
    function transfer(address, uint256) external;
    function balanceOf(address) external view returns(uint);
    function decimals() external view returns (uint8);
    function approve(address, uint) external;
}
```

Recommendation

Switch to use the standard ERC20 interface and ensure the correct return types are returned.

Resolution

RESOLVED

```
interface ERC20{
    function transferFrom(address, address, uint256) external
returns(bool);
    function transfer(address, uint256) external returns(bool);
    function balanceOf(address) external view returns(uint);
    function decimals() external view returns(uint8);
    function approve(address, uint) external returns(bool);
    function totalSupply() external view returns (uint256);
}
```

07

MEDIUM SEVERITY

Function: transferFrom requires approval on msg.sender instead of _from causing interoperability problems with other contracts or wallets

Description

This would cause transactions to fail if _from == msg.sender

```
require(allowed[_from][msg.sender] >= _value, "insufficent approval");
```

Recommendation

Only do this check if the _from != msg.sender

Resolution

RESOLVED

Check implemented.

```
if(_from != msg.sender){
  require(allowed[_from][msg.sender] >= _value, "insufficient
approval");
  allowed[_from][msg.sender] -= _value;
}

require(balanceOf(_from) >= _value, "Insufficient token balance.");
```

08

LOW SEVERITY

Function: transferFrom #156-205 performs a multiplication on the result of a division.

Description

Loss of precision in requires check

```
require(balances[_to] <= maxWalletPercent*(totalSupply/100)
```

Recommendation

Switch to multiplication first.

Resolution

ACKNOWLEDGED

09

MEDIUM SEVERITY

Functions: Multiple - spurious transfer events will misreport activity to scanning and reporting systems.

Description

Example:

```
function ProcessBuyReflection(uint _value, address _payee) internal
returns(uint){

  uint fee = ReflectBuyFeePercent*(_value/100);
  rebaseMult += totalSupply/((totalSupply-fee)*1e18);
  emit Transfer(_payee, address(this), fee);

  return fee;
}
```

Recommendation

Remove all transfer events where no transfer occurs.

Resolution

RESOLVED

All spurious events removed.

10

INFORMATIONAL

mixedCase variable and function names are used throughout.

Description

Variable and function naming throughout are not using consistent mixed case format.

Recommendation

Use mixed case standards throughout and indicate variable names and purposes in comments.

Resolution

ACKNOWLEDGED

Graph

This report has been prepared for Hadouken. Veracity provides an examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

Privileged Roles

The following functions can be called by the admin of the Graph:

- SetBaseContract

The following functions can be called by the base contract of the Graph:

- sweepToken

Issues & Recommendations

11

INFORMATIONAL

mixedCase variable and function names are used throughout.

Description

Variable and function naming throughout are not using consistent mixed case format.

Recommendation

Use mixed case standards throughout and indicate variable names and purposes in comments.

Resolution

ACKNOWLEDGED

12

HIGH SEVERITY

Function: initialize is public without a guard meaning anyone can take ownership of the graph contract.

Description

The initialize function is open to public call and the address is publicly available in the base contract.

```
function initalize(address _admin, address basecontract)
public{
  admin = _admin;
  base = BaseContract(basecontract);
}
```

This means anyone can set the base address and make themselves admin resulting in stolen funds.

Recommendation

Add a one time initialization check to prevent re-initialization.

Resolution

RESOLVED

```
constructor(){
    inital = msg.sender;
}

function initalize(address _admin, address basecontract)
public {
    require(msg.sender == inital, "!initial");

    admin = _admin;
    base = BaseContract(basecontract);
};
```