HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: JulProtocol
Date: October 10, 2020

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Jul Protocol
Type	Token
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Archive Name	backend-liquidity-protocol-master.zip
Commit	C056B183DF2666FA618F465485967A7019FF4103
Timeline	29TH SEP 2020 - 30TH SEP 2020
Changelog	30TH SEP 2020 - Initial Audit 05th OCT 2020 - Second Audit 10th OCT 2020 - Third Audit

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Jul Protocol (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contract and its code review conducted between:

- September 29th, 2020 - September 30th, 2020
- The second review conducted on October 5th, 2020
- The third review conducted on October 10th, 2020

Scope

The scope of the project is smart contracts in the repository:

```
Audit Archive File - backend-liquidity-protocol-master.zip
Commit - c056b183df2666fa618f465485967a7019ff4103
```

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

According to the assessment, the Customer's smart contracts are secure and can be deployed to the mainnet. Though, gas consumption of some functions is not optimal.

Insecure	Poor secured	Secured	Well-secured

You are here

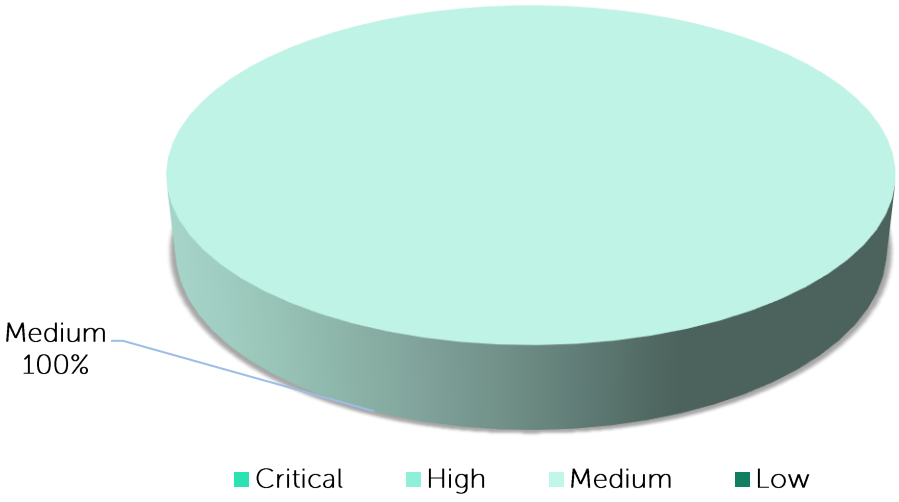
Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section and all found issues can be found in the Audit overview section.

Security engineers found 3 critical, 1 high, 1 medium and 4 low severity issues during the audit. There're no unit tests and deployment scripts. The overall quality of the provided smart contracts is low and requires fixes.

After the second audit security engineers found 1 more high severity issue. 1 critical issue has not been fixed.

After the third audit security engineers found 1 medium severity issue. Also, the code is lack of unit tests. We recommend to cover as much test cases as possible.

Graph 1. The distribution of vulnerabilities after the secondary audit.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets lose or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets lose or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

AS-IS overview

UniswapV2Library.sol

Description

UniswapV2Library is an exact copy of a library provided by the Uniswap and is verified by the community. Link: https://github.com/Uniswap/uniswap-v2-periphery/blob/master/contracts/libraries/UniswapV2Library.sol

JulProtocol.sol

Description

JulProtocol is a token with possibility to perform some operations with the Uniswap.
Secondary audit:
Jul protocol is a proxy between the Jul token and the Uniswap.

Imports

JulProtocol contract has following imports:

- ERC20 - from the OpenZeppelin.
- IERC20 - from the OpenZeppelin.
- IUniswapV2Router02 - from the Uniswap.
- UniswapV2Library - from the project.

Inheritance

JulProtocol is ERC20.

Usages

JulProtocol contract has no usages.

Structs

JulProtocol contract has following data structures:

- struct UserDeposits - stores deposits info.
- struct TotalDeposits - never used.

Enums

- *JulProtocol* contract has no custom enums.

Events

- *JulProtocol* contract has no custom events.

Modifiers

- *JulProtocol* contract has no custom modifiers.

Fields

JulProtocol contract has following fields and constants:

- *uint256 constant TIME_LIMIT = 86400;* - deposits bonus round length.
- *IUniswapV2Router02 public uniswapRouter02;* - uniswap router contract.
- *address router02Address = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;* - uniswap router address.
- *address public UNISWAP_FACTORY = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;* - uniswap factory address.
- *address private WETH;* - uniswap WETH address.
- *uint userCount = 1;* - count users who deposits.
  Removed before the third audit.
- *uint totalEthFee;* - total collected fees.
- *address owner;* - the contract owner.
  Removed before the secondary audit.
- *uint constant teamRelease = 7500000000000000000000;* - tokens allocated for team.
  Removed before the third audit.
- *uint constant marketingRelease = 1250000000000000000000;* - tokens allocated for marketing.
  Removed before the third audit.
- *mapping(address => UserDeposits[]) public protocolusers;* - deposits.
- *mapping(address => TotalDeposits) public usersredeemable;* - never used.
  Removed before the secondary audit.
- *address TOKEN;* - the Jul token address.
  Added before the secondary audit.
- *IERC20 JulToken;* - the Jul token contract.
  Added before the secondary audit.
- *bool release1 = false;* - first tokens release flag.
  Added before the third audit.
- *bool release2 = false;* - second tokens release flag.
  Added before the third audit.
- *bool release3 = false;* - third tokens release flag.
  Added before the third audit.
- *bool release4 = false;* - fourth tokens release flag.
  Added before the third audit.

Functions

JulProtocol has following functions:

- *constructor*
  Description
  Initializes contract values.
  Visibility
  public
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
  None
- *calculatePercent*
  Description
  Calculates percent.
  Visibility
  public pure
  Input parameters
  - uint _eth - sum.
  - uint _percent - percent multiplied by 10.

  Constraints
  None
  Events emit
  None
  Output
  - interestAmt - 0.1 of provided percent of a provided value.
- *calculateInterest (third audit)*
  Description
  Recalculates ETH bonus.
  Visibility
  private
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
  None
- *addEth (initial and second audits)*
  Description

Send any amount of ETH and transfer any amount of tokens to the Uniswap pool on behalf of the contract. Acts like deposit

Visibility

public payable

Input parameters

- *uint _tokenAmount* - amount of tokens to send.
- *uint _ethAmount* - amount of ETH to send.

Constraints

- *_ethAmount* should be more than 1000 wei.

Events emit

None

Output

- amountToken
- amountETH
- liquidity

- ***addEth (third audit)***

Description

Deposits ETH and along with JUL tokens transfers it to the Uniswap pool on behalf of the contract. Acts like deposit.

Visibility

public payable

Input parameters

None

Constraints

- Minimum sum is 0.5 ETH.

Events emit

None

Output

- amountToken
- amountETH
- liquidity

- ***removeLiquidity***

Description

Send `*amt*` of ETH from the uniswap pool to the contract and send tokens instead.

Visibility

public

Input parameters

- *uint256 amt* - amount to send.

Constraints

- Uniswap pool should have corresponding amount of ETH on the contract balance.
- The contract should have `*amt*` of tokens on its balance.

Events emit

None

Output

- *uint256 amountToken*
- *uint256 amountETH*

- ***readUsersDetails***

Description

Get total deposited tokens by the user. And total tokens that is possible to withdraw.

Visibility

public view

Input parameters

- *address _user* - user address.

Constraints

None

Events emit

None

Output

- *uint td*
- *uint trd*
- *uint trwi*

- ***removeEth***

Description

Sens `_amount` of ETH back to a user and redeems his deposits.

Visibility

public payable

Input parameters

- uint256 _amount - amount of ETH to redeem from a user deposit.

Constraints

- *`_amount`* should be less or equal to *`trwi`*.
- The contract should have enough ETH on its balance.

Events emit

None

Output

- *uint td*
- *uint trd*
- *uint trwi*
- bool status

- ***getLiquidityBalance***

Description

Get liquidity balance from the Uniwsap.

Visibility

public view

Input parameters

None

Constraints

None

Events emit

None
Output
- uint256 liquidity

- ***teamFromInt (removed before the third audit)***
  Description
  Send 1% of collected fees to the contract owner.
  Visibility
  external
  Input parameters
  None
  Constraints
  - Can only be called by the contract owner.

  Events emit
  None
  Output
  None

- ***tokenRelease (initial and second audits)***
  Description
  Trying to send `*teamRelease*`+`*marketingRelease*` amount of ETH from the contract to the contract owner.
  Visibility
  external
  Input parameters
  - uint amount - never used.

  Constraints
  - The contract should have at least `*teamRelease*`+`*marketingRelease*` amount of ETH.

  Events emit
  None
  Output
  None

- ***tokenRelease (third audit)***
  Description
  Transfers `*lockedTokens*` amount of tokens if Uniwswap ETH balance milestones are reached. 4 transfers are allowed with following milestones: 6000 ETH, 15000 ETH, 30000 ETH and 45000 ETH.
  Visibility
  external
  Input parameters
  None
  Constraints
  - Can only be called by the contract owner.
  - The contract should have at least `*lockedTokens*` amount of ETH.

  Events emit

None
Output
None

- *transferOwnership (removed after the initial audit)*
  Description
  Transfer ownership to a specified address.
  Visibility
  public
  Input parameters
    - address _newOwner - new owner address.
  Constraints
    - Can only be called by the owner.
  Events emit
  None
  Output
  None
- *protocolBalance*
  Description
  Get current JUL balance of the contract.
  Visibility
  public view
  Input parameters
  None
  Constraints
  Non
  Events emit
  None
  Output
    - uint - current JUL balance of the contract.
- *getBalanceFromUniswap*
  Description
  Get reserves from the Uniswap.
  Visibility
  public view
  Input parameters
  None
  Constraints
  Non
  Events emit
  None
  Output
    - uint reserveA
    - uint reserveB

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

Critical

1. `*tokenRelease*` function transfers ETH instead of tokens.

   Fixed before the secondary audit.

2. It's possible to pass any `*_tokenAmount*` value to the `*addEth*` function. The only limit is 100000000000000000000000 that is minted to the contract address during the contract construction.

   NOT(!) fixed before the secondary audit.

   Fixed before the third audit.

3. `*removeLiquidity*` function can be called by anyone without any validations. It can lead to inconsistency of the pool.

   Fixed before the secondary audit.

High

1. `*addEth*` and `*removeEth*` functions has redundant fee transfers to the contract itself.

   Partially fixed before the secondary audit.

   Fixed before the third audit.

2. Before the secondary audit code of the Jul token has been removed from the Customer did no provide a code. It can lead to possible token supply manipulation or other security leaks.

   Fixed before the third audit.

Medium

1. Instead of manual access control managing, it's recommended to use OpenZeppelin `*Ownable*` contract.

   Fixed before the secondary audit.

2. The removeETH receives ETH to the contract and then transfers it to a user. This behavior lids to increased gas consumption.

Low

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

1. `*buyJulFromFee*` function has empty body and should be removed.

   Fixed before the secondary audit.

2. `*readUsersDetails*` function has unused `*depositsCount*` variable.

   Not fixed before the secondary audit.

   Fixed before the third audit.

3. `*tokenRelease*` function has unused argument `*amount*`, and unused local variable `*totalBal*`.

   Fixed before the secondary audit.

4. `*usersredeemable*` field is never used and can be removed.

   Fixed before the secondary audit.

5. `*TotalDeposits*` data structure is never used and can be removed.

   Fixed before the secondary audit.

Lowest / Code style / Best Practice

1. A lot of code style issues were found by static code analyzers.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Conclusion

Smart contracts within the scope was manually reviewed and analyzed with static analysis tools. For the contract high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found 3 critical, 1 high, 1 medium and 4 low severity issues during the audit. There are no unit tests and deployment scripts. The overall quality of the provided smart contracts is low and requires fixes.

After the secondary audit security engineers found 1 more high severity issue. 1 critical issue has not been fixed.

After the third audit security engineers found 1 medium severity issue. Also, the code is lack of unit tests. We recommend to cover as much test cases as possible.

Violations in following categories were found and addressed to Customer:

Category	Check Item	Comments
Code review	▪ Repository consistency	▪ The code is lack of unit tests and deployment scripts.
	▪ Increased gas consumption	▪ Gas usage is not optimal.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.