HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: TrustSwap
Date: September 7th, 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 TrustSwap
Type	Token swap gateway
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Files Name	trustswap-sol-pdf.zip, SwapStakingContract.sol
SHA256 hashs	5cc6bebf93d1e5e8edf7a707db97f452c9625f2cfdce70b6a3521d03fd29b4d5, 13c0ec2992ebe8ac1a6eb545df9a8d19d3978e5b87ab34dcdc104fc5f20f54e5
Timeline	AUGUST 31ST, 2020 - SEP 07TH, 2020
Changelog	03RD SEP 2020 - Initial Audit 07TH SEP 2020 - Added SwapStakingContract.sol Contract 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 TrustSwap (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer`s smart contract and its code review conducted between August 31st, 2020 - September 7th, 2020.

Scope

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

Audit Files	
trustswap-sol-pdf.zip	
IERC20Extended.sol IPriceEstimator.sol PriceEstimator.sol SwapPaymentScheduler.sol	
SHA256	5cc6bebf93d1e5e8edf7a707db97f452c9625f2cfdce70b6a3521d03fd29b4d5
SwapStakingContract.sol	
SHA256	13c0ec2992ebe8ac1a6eb545df9a8d19d3978e5b87ab34dcdc104fc5f20f54e5

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

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

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts do have one high vulnerability that needs to be fixed. Though, some minor fixes are still required.

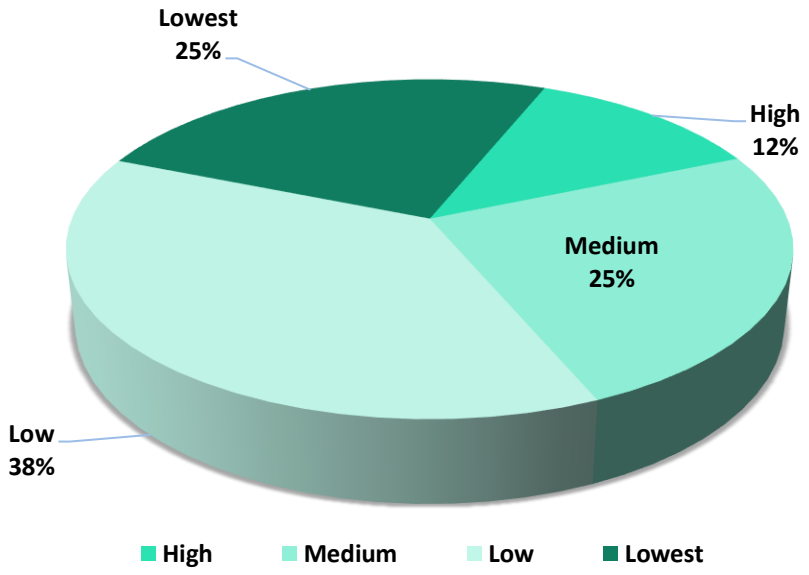
Insecure	Poor secured	Secured	Well-secured

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 essential 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.

During the audit, we found 1 high, 2 medium, 3 low, and 2 lowest severity issues and a bunch of code style issues.

Graph 1. The distribution of vulnerabilities.



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

IERC20Extended.sol

IERC20Extended is an interface, defines 2 functions:

- *decimals*
- *burnFrom*

IPriceEstimator.sol

IPriceEstimator is an interface, defines 2 functions:

- *getEstimatedETHforERC20*
- *getEstimatedERC20forETH*

PriceEstimator.sol

PriceEstimator imports:

- **Ownable** from *OpenZeppelin*
- **Address** from *OpenZeppelin*
- **IUniswapV2Router02** from *Uniswap*
- **IPriceEstimator**

Contract **PriceEstimator** is *IPriceEstimator*, *Initializable* and *OwnableUpgradeSafe.* It using **Address** for *address* type.

PriceEstimator has 1 field: *uniswapRouter.*

Contract **PriceEstimator** has 1 modifier: *onlyContract* - check if address is contract.

PriceEstimator has 7 functions:

- *initialize* - an external function used to initialize contract by setting the *uniswapRouter* field.
- *__PriceEstimator_init* - an internal function used to initialize contract by setting the *uniswapRouter* field.
- *setUniswapRouter* - an external function with *onlyOwner* and *onlyContract* modifiers. Used to set *uniswapRouter* field.
- *getEstimatedETHforERC20* - an external view function used to calculate the minimum amount of ETH required to receive a certain amount of tokens.

- *getPathForETHtoERC20* - an internal view function used to fetch path from ETH to token address from *uniswapRouter*.
- *getEstimatedERC20forETH* - an external view function used to calculate the minimum amount of tokens required to receive a certain amount of ETH.
- *getPathForERC20toETH* - an internal view function used to fetch path from token address to ETH from *uniswapRouter*.

SwapPaymentScheduler.sol

SwapPaymentScheduler imports:

- **SafeMath** from *OpenZeppelin*
- **Pausable** from *OpenZeppelin*
- **SafeERC20** from *OpenZeppelin*
- **Ownable** from *OpenZeppelin*
- **ERC20** from *OpenZeppelin*
- **IERC20Extended**
- **IPriceEstimator**

Contract **SwapPaymentScheduler** is *Initializable, OwnableUpgradeSafe and PausableUpgradeSafe.* It using **Address** for *address* type, **SafeERC20** for *IERC20* type and **SafeMath** for *uint256* type.

SwapPaymentScheduler has 1 enum:

- *enum Status { CLOSED, OPEN }*

Contract **SwapPaymentScheduler** has 1 data struct:

- *Payment* - used to describe a payment itself. Defines 7 fields:
  - *address token* - token address.
  - *address sender* - sender address.
  - *address payable beneficiary* - address of the beneficiary.
  - *uint256 amount* - amount of tokens to pay.
  - *uint256 releaseTime* - time in seconds after which tokens to be released.

- *uint256 createdAt* - payment creation timestamp.
- *Status status* - payment status, can be *CLOSED* (by default) or *OPEN*.

Contract **SwapPaymentScheduler** has 11 fields and constants:

- *IERC20 private _swapToken* - swap token contract.
- *address payable private _feesWallet* - wallet where fees will go.
- *address private _devWallet* - wallet where dev fund will go.
- *address constant private ETH_ADDRESS* - constant for ETH address.
- *uint256 private _paymentId* - global payment id. Also it gives total number of payments made so far.
- *mapping(address => uint256[]) private _beneficiaryVsPaymentIds* - list of all payment ids for a user/beneficiary.
- *mapping(address => uint256[]) private _senderVsPaymentIds* - list of all payment ids for a sender.
- *mapping(uint256 => Payment) private _idVsPayment* - payments with id
- *IPriceEstimator private _priceEstimator* - PriceEstimator contract.
- *uint256 private _ethFeePercentage* - ETH percentage fee.
- *uint256 private _allowedFeeSlippagePercentage* - allowed fee slippage percentage for fee in ERC20.

Contract **SwapPaymentScheduler** defines 5 events:

- *event PaymentScheduled(address indexed token, address indexed sender, address indexed beneficiary, uint256 id, uint256 amount, uint256 releaseTime, uint256 fee, bool isFeeInSwap, bool calcFeeUsingTotalSupply);*
- *event PaymentReleased(uint256 indexed id, address indexed beneficiary, address indexed token);*
- *event FeeWalletChanged(address indexed wallet);*

- *event DevWalletChanged(address indexed wallet);*
- *event SwapTokenUpdated(address indexed swapTokenAddress);*

Contract **SwapPaymentScheduler** has 2 modifiers:

- *onlyContract* - check if address is contract.
- *canRelease* - check if payment can be released.

SwapPaymentScheduler has 34 functions:

- *initialize* - an external function used to set *_swapToken, _feesWallet, _devWallet, _priceEstimator, _ethFeePercentage, _allowedFeeSlippagePercentage* fields.
- *__SwapPaymentScheduler_init* - an internal initializer function used to set *_swapToken, _feesWallet, _devWallet, _priceEstimator, _ethFeePercentage, _allowedFeeSlippagePercentage* fields.
- *__SwapPaymentScheduler_init_unchained* - an internal initializer function used to set *_swapToken, _feesWallet, _devWallet, _priceEstimator, _ethFeePercentage, _allowedFeeSlippagePercentage* fields.
- *getFeesWallet* - an external view function used to get the fee receiver wallet address.
- *getDevWallet* - an external view function used to get the dev fund wallet address.
- *getSwapToken* - an external view function used to get the swap token address.
- *getPriceEstimator* - an external view function used to get the address of PriceEstimator contract.
- *getPaymentDetails* - an external view function used to get payment details.
- *getBeneficiaryPaymentIds* - an external view function used to get all payment ids of the beneficiary.
- *getSenderPaymentIds* - an external view function used to get all payment ids for sender.
- *setSwapToken* - an external function with the *onlyOwner* and *onlyContract* modifiers. Used by Owner to set swap token address.

- *setFeeWallet* - an external function with the *onlyOwner* modifier. Used by Owner to set fee receiver wallet address.
- *setDevWallet* - an external function with the *onlyOwner* modifier. Used by Owner to set dev fund wallet address.
- *setPriceEstimator* - an external function with the *onlyOwner* and *onlyContract* modifiers. Used by Owner to set PriceEstimator contract address.
- *setEthFeePercentage* - an external function with the *onlyOwner* modifier. Used by Owner to ETH fee percent.
- *setAllowedFeeSlippagePercentage* - an external function with the *onlyOwner* modifier. Used by Owner to set allowed fee slippage percentage for fee in ERC20.
- *getFeeInEthForEth* - a public view function used to get fee in ETH for ETH.
- *getFeeInEthForERC20* - a public view function used to get fee in ETH for ERC20.
- *getFeeInEthForERC20UsingTotalSupply* - a public view function used to get fee in ETH for ERC20 using total supply.
- *getFeeInSwapForETH* - a public view function used to get fee in swap for ETH.
- *getFeeInSwapForERC20* - a public view function used to get fee in swap for ERC20.
- *_getEquivSwapFee* - a private view function used to get fee in swap for ETH.
- *schedulePayment* - an external payable function with whenNotPaused modifier. Used to schedule payment.
- *_schedulePayment* - a private function used to schedule payment.
- *_scheduleETH* - a private function used to schedule payment in ETH.
- *_scheduleERC20* - a private function with *onlyContract* modifier. Used to schedule in ERC20 tokens.
- *_distributeFees* - a private function used to distribute fee.

- *scheduleBulkPayment* - an external payable function with whenNotPaused modifier. Used to schedule multiple payments.
- *release* - an external function with *canRelease* modifier. Allows beneficiary of payment to release payment after release time.
- *releasable* - a public view function used to check whether payment can be released or not.
- *_releaseETH* - a private function used to release ETH.
- *_releaseERC20* - a private function used to release ERC20.
- *pause* - an external function with *onlyOwner* modifier. Used by Owner to trigger stopped state.
- *unpause* - an external function with *onlyOwner* modifier. Used by Owner to return to normal state.

SwapStakingContract.sol imports *SafeMath, Math, Address, Arrays, ReentrancyGuard, Pausable, ERC20/IERC20, AccessControl* from OpenZepplin.

SwapStakingContract using:

- *SafeMath* for uint256;
- *Math* for uint256;
- *Address* for address;
- *Arrays* for uint256[];

SwapStakingContract defines 1 struct *StakeDeposit* that has 6 fields:

- *amount* - a stake deposit amount;
- *startDate* - a stake deposit start date;
- *endDate* - a stake deposit end date;
- *entryRewardPoints* - a reward points at the start of the stake deposit;
- *exitRewardPoints* - a reward points at the end of the stake deposit;

- *exists* - a stake deposit existence flag;

SwapStakingContract defines 13 constants and fields:

- *PAUSER_ROLE* - a constant defines role that can to pause;
- *OWNER_ROLE* - a constant defines owner role;
- *REWARDS_DISTRIBUTOR_ROLE* - a constant defines rewards distributor role;
- *token* - an IERC20 token contract;
- *rewardsAddress* - an address where the reward is stored;
- *maxStakingAmount* - a maximum staking amount;
- *currentTotalStake* - a current staking amount;
- *unstakingPeriod* - a number of days required for withdrawal;
- *totalRewardPoints* - a total reward points;
- *rewardsDistributed* - an amount of rewards distributed;
- *rewardsWithdrawn* - an amount of rewards withdrawn;
- *totalRewardsDistributed* - a sum of rewards distributed;
- *_stakeDeposits* - a mapping to store stake deposit by address;

SwapStakingContract has 4 events:

event StakeDeposited(address indexed account, uint256 amount);

event WithdrawInitiated(address indexed account, uint256 amount);

event WithdrawExecuted(address indexed account, uint256 amount, uint256 reward);

event RewardsDistributed(uint256 amount);

Contract **SwapStakingContract** has 2 modifiers:

guardMaxStakingLimit - checks if the staking limit has been exceeded when trying to stake more tokens;

onlyContract - checks if address is contract;

SwapStakingContract has 15 functions:

- *initialize* - a public function with *onlyContract(token)* modifier. Used to set *token*, *rewardsAddress*, *maxStakingAmount*, *unstakingPeriod*;
- *__SwapStakingContract_init* - an internal initializer function used to set *token*, *rewardsAddress*, *maxStakingAmount*, *unstakingPeriod*;
- *__SwapStakingContract_init_unchained* - an internal initializer function used to set roles;
- *pause* - a public function used to trigger stopped state;
- *unpause* - a public function used to return to normal state;
- *setRewardAddress* - a public function with *whenPaused* modifier. Used to set rewards address;
- *setTokenAddress* - an external function with *onlyContract* and *whenPaused* modifiers. Used to set token contract;
- *deposit* - a public function with *nonReentrant*, *whenNotPaused*, *guardMaxStakingLimit(amount)* modifiers. Used to deposit an amount of tokens;
- *initiateWithdrawal* - an external function with *nonReentrant* and *whenNotPaused* modifiers. Used to initiate withdrawal;
- *executeWithdrawal* - an external function with *nonReentrant* and *whenNotPaused* modifiers. Used to execute withdrawal;
- *getStakeDetails* - an external view function used to get stake details;
- *_computeReward* - a private view function used to compute reward for staking;
- *distributeRewards* - an external function with *nonReentrant* and *whenNotPaused* modifiers. Used to distribute rewards;

- *_distributeRewards* - a private function with *whenNotPaused* modifier. Used to distribute rewards;
- *version* - a public pure function used to get contract version;

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

Audit overview

Critical

No critical issues were found.

High

1. Function *deposit* should be *external*. If another function calls the *nonReentrant* function it is no longer protected.

Medium

1. Function *setEthFeePercentage* has no validation of *ethFeePercentage* parameter. All calculations using *_ethFeePercentage* expect it to be in the range 0..100.
2. Function *setAllowedFeeSlippagePercentage* has no validation of *allowedFeeSlippagePercentage* parameter. All calculations that are using *_allowedFeeSlippagePercentage* expect it to be in the range 0..100.

Low

1. Missing Uniswap router address validation at *__PriceEstimator_init* function.
2. Checking for the length of arrays does not guarantee data consistency. It is recommended to use data structures as input parameters.
3. Missing zero address validations for token at *__SwapStakingContract_init* and *setTokenAddress* functions.

Lowest / Code style / Best Practice

1. Functions *getFeeInEthForERC20*, *getFeeInEthForERC20UsingTotalSupply*, *_getEquivSwapFee*, *_distributeFees* have a lot of “magic” numbers that should be moved to named constants (e.g. uniswap 0.30% fees).
2. *Pausable.sol* imported twice at *SwapPaymentScheduler*.sol.

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.

Violations in following categories were found and addressed to Customer:

Category	Check Item	Comments
Code review	▪ Reentrancy ▪ Repository Consistency	▪ Broken environment ▪ Lack of deployment scripts

Security engineers found 1 high, 2 medium, 3 low and 2 lowest severity issues during audit. It’s recommended to fix all those issues.

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 the 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 the 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.