HashEx

Trader Joe

smart contracts audit report

Prepared for:

traderjoexyz.com

Authors: HashEx audit team

September 2021

Contents

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Introduction

HashEx was commissioned by the TraderJoe team to perform an audit of their smart contracts. The audit was conducted between July 27 and August 01, 2021.

The code located in @traderjoe-xyz/joe-core GitHub repository was audited after beda9cf commit. Repository contains tests for major contracts, but not for DEX contracts nor for the JoeHatToken. Code was provided with brief descriptions of the contracts. MasterChefJoeV2 contract has its own documentation. Team website has the docs section.

The same contracts are deployed to the Avalanche C-Chain:

JoeToken	0x6e84a6216eA6dACC71eE8E6b0a5B7322EEbC0fDd,
JoeBar	0x57319d41F71E81F3c65F2a47CA4e001EbAFd4F33,
JoeFactory	0x9Ad6C38BE94206cA50bb0d90783181662f0Cfa10,
JoeRouter02	0x60aE616a2155Ee3d9A68541Ba4544862310933d4,
MasterChefJoeV2	0xd6a4F121CA35509aF06A0Be99093d08462f53052,
JoeHatToken	0x82FE038Ea4b50f9C957da326C412ebd73462077C,
Cliff	0xaFF90532E2937fF290009521e7e120ed062d4F34,
Cliff	0xc13B1C927565C5AF8fcaF9eF7387172c447f6796.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts.
- Formally check the logic behind given smart contracts.

Information in this report should be used to understand the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The contracts are mostly forks of SushiSwap and for this reason we focused on the unaudited parts of code, as well as modifications made by the TraderJoe team.

Update: TraderJoe team has responded to this report. Individual responses were added after each item in the section.

Contracts overview

`libraries/BoringERC20.sol, SafeERC20.sol, SafeMath.sol;`
`boringcrypto/BoringOwnable.sol`

Support contracts from BoringCrypto [1] and OpenZeppelin.

`rewarders/ SimpleRewarderPerSec.sol`

Rewarder contract (see docs) for MasterChefs interaction.

`traderjoe/ JoeERC20.sol, JoeFactory.sol, JoePair.sol, JoeRouter02.sol, libraries`

Similar to UniswapV2 contracts (audit available [7]).

`Cliff.sol`

Locker contract for fixed ERC20 token.

`JoeBar.sol`

Similar to SushiBar by SushiSwap, used for swapping JOE and xJOE tokens.

`JoeHatToken.sol`

Burnable ERC20 [8] token with initial supply of `150*(10**decimals)`.

`JoeMaker.sol`

Similar to SushiMaker by SushiSwap, used for swapping tokens with JoePair contracts.

`JoeRoll.sol`

Similar to SushiRoll by SushiSwap, used for liquidity migration.

`JoeToken.sol`

Similar to SushiToken by SushiSwap, ERC20 standard [8] token with governance from YAM/Compound.

`MasterChefJoeV2.sol`

MasterChef contract modified to interact with other MasterChefs via Rewarder contracts (e.g. MasterChefRewarderPerBlock).

`Zap.sol`

Similar to Zap contract by PancakeBunny, used for swapping tokens, LP tokens and system currency.

Found issues

ID	Title	Severity	Response
01	MasterChefJoeV2: fees are not limited	High	Fixed
02	MasterChefJoeV2: emission rate not limited	High	Fixed
03	JoeToken: delegates not transferred	High	Acknowledged
04	MasterChefJoeV2: massUpdatePools() gas	Medium	Acknowledged
05	MasterChefJoeV2: pool token support	Medium	Acknowledged
06	JoeToken: minting capped token	Medium	Acknowledged
07	SimpleRewarderPerSec: rewarding model	Medium	Acknowledged
08	Zap: sweep() may exceed gas limit	Medium	Acknowledged
09	Zap: no checks for slippage and deadline	Medium	Acknowledged
10	MasterChefJoeV2: checks uint>=0	Low	Acknowledged
11	MasterChefJoeV2: Rewarder tokens not checked	Low	Acknowledged
12	MasterChefJoeV2: excessive computations	Low	Acknowledged
13	MasterChefJoeV2: console import	Low	Acknowledged
14	SimpleRewarderPerSec: inconsistent comments	Low	Acknowledged
15	SimpleRewarderPerSec & MasterChefJoeV2: experimental features	Low	Acknowledged
16	JoeLibrary: inconsistent comment	Low	Acknowledged
17	Cliff: library not used	Low	Acknowledged
18	Cliff: input parameters not checked	Low	Acknowledged
19	Cliff: locked third-party tokens	Low	Acknowledged
20	JoeMaker: inconsistent comments	Low	Acknowledged
21	JoeMaker: excessive computations	Low	Acknowledged

22	JoeMaker: convertMultiple() arrays length	Low	Acknowledged
23	General recommendations	Low	Acknowledged

#01 MasterChefJoeV2: fees are not limited — High

Set functions `changeDevPercent()`, `setTreasuryPercent()` and `setInvestorPercent()` L418-464 update fees without filtering the input data. The owner can use this function to take 100% of rewards.

Recommendation: limit the fees sum from above. We recommend checking input data in the constructor/initializer too.

Update: the issue was fixed by transferring ownership to a custom Timelock contract with the parameters filtered by signature and capped values of the fees.

#02 MasterChefJoeV2: emission rate not limited — High

`updateEmissionRate()` function L468 is used for updating the `joePerSec` parameter. Although it's the onlyOwner function, we recommend adding safety guards — capping the new value. If the owner's account gets compromised or the owner acts maliciously, the attacker can set an arbitrary big value for the `joePerSec` variable. In such a case the number of tokens till cap will be minted soon and the token price will drop.

Recommendation: add requirements for input data in `updateEmissionRate()`.

Update: the issue was fixed by transferring ownership to a custom Timelock contract with the parameters filtered by signature and capped value of the emission rate.

#03 JoeToken: delegates not transferred — High

`_moveDelegates()` function in L203 of JoeToken contract is designed to be used with every token transfer. However, in JoeToken it's called with minting new tokens, but not with the usual transfers. The origin (SushiToken) has a warning (L8) about this issue. Issue allows to mint any number of delegation votes.

Recommendation: stick with the original governance logic and move delegates with transfers.

#04 MasterChefJoeV2: massUpdatePools() gas — Medium

Unlimited length of the `poolInfo[]` array may block the `massUpdatePools()` calls in `add()`, `set()` and `updateEmissionRate()` public functions.

#05 MasterChefJoeV2: pool token support — Medium

`lpToken` parameter of pools in the MasterChefJoeV2 contract intend to be used with LP tokens, i.e. ERC20 tokens generated with SwapPair contracts. Although tokens with transfer commission are ERC20 in most cases, they aren't supported by the original MasterChef as the real transferred amount isn't checked in the `deposit()` function.

#06 JoeToken: minting capped token — Medium

Capped ERC20 JoeToken is meant to be used with MasterChef contracts, so once the cap is reached the chef contract will fail on minting, and reward withdrawals will be unavailable.

#07 SimpleRewarderPerSec: rewarding model — Medium

Fair rewards are possible only in case of uniform distribution of second rewards in SimpleRewarder contracts. Otherwise may update their `rewardDebt` but receive less than the others or nothing at all.

#08 Zap: sweep() may exceed gas limit — Medium

With the growing of the `tokens[]` array `sweep()` function may become uncallable due to block gas limit L384.

#09 Zap: no checks for slippage and deadline — Medium

The zap contract is a helper contract to swap tokens. It uses the JoeRouterV2 contract (fork of UniwapRouterV2 contract) to do the conversion, but it calls the router's functions with 100% slippage and no deadline set. The transactions sent from this contract may be frontrun resulting in swaps with an undesired rate (sandwich attacks). Also if a transaction is sent with a small gas price it can be mined for a long time resulting in swaps with a significantly changed rate against the moment the transaction was added to the block.

#10 MasterChefJoeV2: checks uint>=0 — Low

Both MasterChefJoe contracts perform checks on input uint256 parameters to be greater or equal zero.

#11 MasterChefJoeV2: Rewarder tokens not checked — Low

Adding or updating address of Rewarder contract to the pool of MasterChefJoeV2 should also check the rewardToken and/or lpToken of Rewarder contract.

#12 MasterChefJoeV2: excessive computations — Low

Check for duplication in `add()` function L186 should require the success of `EnumerableSet.add()` method.

#13 MasterChefJoeV2: console import — Low

Import of the Hardhat console should be removed.

#14 SimpleRewarderPerSec: inconsistent comments — Low

All 4 Rewarders contracts contain the interface of IMasterChefJoeV2 with comments from SUSHI. MasterRewarderPerBlock's comment L38 is about block number but placed near `lastRewardTimestamp` variable.

#15 SimpleRewarderPerSec & MasterChefJoeV2: experimental features — Low

We recommend avoiding experimental features of the Solidity compiler.

#16 JoeLibrary: inconsistent comment — Low

Comment in L39 of the JoeLibrary contract was removed in the deployed JoeRouter02 contract.

#17 Cliff: library not used — Low

Math.sol library is imported in the Cliff contract but never used.

#18 Cliff: input parameters not checked — Low

Input parameters in the constructor of the Cliff contract aren't checked. Deployed parameters should be checked before locking funds in the Cliff instances. Timestamps should be checked if setted in milliseconds.

#19 Cliff: locked third-party tokens Low

Locker contract usually magnets the third-party ERC20 tokens. We recommend adding recover function that allows to transfer arbitrary ERC20 with `require(recoveredToken != vestedToken)`.

#20 JoeMaker: inconsistent comments Low

Addresses in comments of the JoeMaker contract L24-33 are for Ethereum and SushiSwap contracts. TODO comments in L111, 243, 250.

#21 JoeMaker: excessive computations Low

`_swap()` function of the JoeMaker contract calculates local variable `amountInWithFee` `L236`, but never uses it performing the same multiplication twice later.

#22 JoeMaker: convertMultiple() arrays length Low

`convertMultiple()` function of the JoeMaker contract should check matching of the input arrays' lengths.

#23 General recommendations Low

We recommend avoiding experimental features of the Solidity compiler, e.g. in SimpleRewarderPerSec & MasterChefJoeV2.
MasterChefJoeV2 contract performs excessive checks on fee percents input data in constructor and in separate set functions. Only the sum of all fees should be checked.
Input data in constructors should be checked with require statements where possible.
We recommend using fixed Pragma versions, see Cliff contract.
Cliff contract contains comment 'Optionally revocable by the owner', but it's not even Ownable.

Conclusion

3 high severity issues were found.

Audit includes recommendations on the code improving and preventing potential attacks.

Audited contracts are deployed to Avalanche C-Chain:

JoeToken	0x6e84a6216eA6dACC71eE8E6b0a5B7322EEbC0fDd,
JoeBar	0x57319d41F71E81F3c65F2a47CA4e001EbAFd4F33,
JoeFactory	0x9Ad6C38BE94206cA50bb0d90783181662f0Cfa10,
JoeRouter02	0x60aE616a2155Ee3d9A68541Ba4544862310933d4,
MasterChefJoeV2	0xd6a4F121CA35509aF06A0Be99093d08462f53052,
JoeHatToken	0x82FE038Ea4b50f9C957da326C412ebd73462077C,
Cliff	0xaFF90532E2937fF290009521e7e120ed062d4F34,
Cliff	0xc13B1C927565C5AF8fcaF9eF7387172c447f6796.

Update: TraderJoe team has responded to this report. The Timelock contract was deployed and gained the ownership of the MasterChefJoeV2 to fix 01 and 02 issues. The contract was deployed with the minimum delay of 12 hours and limits of 20%, 20% and 10% for dev, treasury and investor fees. Address of the Timelock contract in Avalanche C-Chain:

Timelock	0xAdaF18d79f316005542da4eCb1624B59c4e6e398.

The JoeToken governance is subject to delegation error, see 03.

Users should pay attention to the possible MasterChefJoeV2's ownership transfer due to low minimal delay of the Timelock contract.

References and forks

1. BoringCrypto GitHub
2. CompoundFinance's Timelock contract
3. Timelock audit by OpenZeppelin
4. SushuSwap's MasterChef contract
5. SushiSwap audit by PeckShield
6. SushiSwap audit by Quantstamp
7. Uniswap audit
8. ERC-20 Token Standard

Appendix A. Issues' severity classification

We consider an issue critical, if it may cause unlimited losses or break the workflow of the contract and could be easily triggered.

High severity issues may lead to limited losses or break interaction with users or other contracts under very specific conditions.

Medium severity issues do not cause the full loss of functionality but break the contract logic.

Low severity issues are typically nonoptimal code, unused variables, errors in messages. Usually, these issues do not need immediate reactions.

Appendix B. List of examined issue types

Business logic overview

Functionality checks

Following best practices

Access control and authorization

Reentrancy attacks

Front-run attacks

DoS with (unexpected) revert

DoS with block gas limit

Transaction-ordering dependence

ERC/BEP and other standards violation

Unchecked math

Implicit visibility levels

Excessive gas usage

Timestamp dependence

Forcibly sending ether to a contract

Weak sources of randomness

Shadowing state variables

Usage of deprecated code