Trader Joe contest
Findings & Analysis Report

2022-03-07

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of Trader Joe contest smart contract system written in Solidity. The code contest took place between January 25—January 27 2022.

Wardens

43 Wardens contributed reports to the Trader Joe contest:

1. cmichel
2. static
3. defsec
4. Dravee
5. sirhashalot
6. jayjonah8
7. kirk-baird
8. robee
9. Jujic
10. pedroais
11. TomFrenchBlockchain
12. hubble (ksk2345 and shri4net)
13. WatchPug (jtp and ming)
14. 0x1f8b
15. pauliax
16. harleythedog
17. cccz
18. Ruhum
19. p4st13r4 (0x69e8 and 0xb4bb4)
20. Funen
21. wuwe1
22. 0v3rf10w
23. UncleGrandpa925

24. Czar102
25. hyh
26. gzeon
27. csanuragjain
28. hack3r-0m
29. Tomio
30. bobi
31. rfa
32. byterocket (pseudorandom and pmerkleplant)
33. saian
34. MetaOxNull
35. 0x0x0x
36. Rhynorater
37. d4rk
38. yeOlde
39. solgryn

This contest was judged by LSDan (ElasticDAO).

Final report assembled by liveactionllama and itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 43 unique vulnerabilities and 117 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity, 15 received a risk rating in the category of MEDIUM severity, and 26 received a risk rating in the category of LOW severity.

C4 analysis also identified 6 non-critical recommendations and 68 gas optimizations.

Scope

The code under review can be found within the C4 Trader Joe contest repository, and is composed of 4 smart contracts written in the Solidity programming language and includes 1111 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (2)

[H-01] Users can lose value in emergency state

Submitted by cmichel, also found by static

Imagine the following sequence of events:

- `LaunchEvent.createPair()` is called which sets `wavaxReserve = 0`, adds liquidity to the pair and receives `lpSupply` LP tokens.
- `LaunchEvent.allowEmergencyWithdraw()` is called which enters emergency / paused mode and disallows normal withdrawals.

- Users can only call `LaunchEvent.emergencyWithdraw` which reverts as the WAVAX reserve was already used to provide liquidity and cannot be paid out. Users don't receive their LP tokens either. The users lost their entire deposit in this case.

Recommendation

Consider paying out LP tokens in `emergencyWithdraw`.

cryptofish7 (Trader Joe) confirmed and commented:

> Fix: https://github.com/traderjoe-xyz/rocket-joe/pull/99

[H-02] Wrong token allocation computation for token decimals != 18 if floor price not reached

Submitted by cmichel

In `LaunchEvent.createPair`, when the floor price is not reached (`floorPrice > wavaxReserve * 1e18 / tokenAllocated`), the tokens to be sent to the pool are lowered to match the raised WAVAX at the floor price.

Note that the `floorPrice` is supposed to have a precision of 18:

> /// @param _floorPrice Price of each token in AVAX, scaled to 1e18

The `floorPrice > (wavaxReserve * 1e18) / tokenAllocated` check is correct but the `tokenAllocated` computation involves the `token` decimals:

```
// @audit should be wavaxReserve * 1e18 / floorPrice
tokenAllocated = (wavaxReserve * 10**token.decimals()) / floorPr
```

This computation does not work for `token`s that don't have 18 decimals.

Example

Assume I want to sell `1.0 wBTC = 1e8 wBTC` (8 decimals) at `2,000.0 AVAX = 2,000 * 1e18 AVAX`. The `floorPrice` is `2000e18 * 1e18 / 1e8 = 2e31`

Assume the Launch event only raised `1,000.0 AVAX` - half of the floor price for the issued token amount of `1.0 WBTC` (it should therefore allocate only half a WBTC) - and the token amount will be reduced as: `floorPrice = 2e31 > 1000e18 * 1e18 / 1e8 = 1e31 = actualPrice`. Then, `tokenAllocated = 1000e18 * 1e8 / 2e31 = 1e29 / 2e31 = 0` and no tokens would be allocated, instead of `0.5 WBTC = 0.5e8 WBTC`.

The computation should be `tokenAllocated = wavaxReserve * 1e18 / floorPrice = 1000e18 * 1e18 / 2e31 = 1e39 / 2e31 = 10e38 / 2e31 = 5e7 = 0.5e8`.

Recommendation

The new `tokenAllocated` computation should be `tokenAllocated = wavaxReserve * 1e18 / floorPrice;`.

cryptofish7 (Trader Joe) confirmed and commented:

> Fix: https://github.com/traderjoe-xyz/rocket-joe/pull/76

Medium Risk Findings (15)

[M-01] Improper Upper Bound Definition on the Fee

Submitted by Jujic

The `rJoePerSec` does not have any upper or lower bounds. Values that are too large will lead to reversions in several critical functions.

Proof of Concept

https://github.com/code-423n4/2022-01-trader-joe/blob/a1579f6453bc4bf9fb0db9c627beaa41135438ed/contracts/RocketJoeSt

aking.sol#L151

```
    function updateEmissionRate(uint256 _rJoePerSec) external onlyOw
        updatePool();
        rJoePerSec = _rJoePerSec;
        emit UpdateEmissionRate(msg.sender, _rJoePerSec);
    }
```

Tools Used

Remix

Recommended Mitigation Steps

Consider define upper and lower bounds on the `_rJoePerSec`.

cryptofish7 (Trader Joe) confirmed, but disagreed with severity and commented:

> Confirming issue but disagree with severity.

> Fix: https://github.com/traderjoe-xyz/rocket-joe/pull/112

[M-02] Owner of LaunchEvent token has the ability to DOS attack the event

Submitted by Ruhum, also found by TomFrenchBlockchain

The owner of the token for which the LaunchEvent was created, has the ability to DOS attack the event. They can prevent the LaunchEvent from creating a JoePair which in turn limits the access to the following two functions: `withdrawLiquidity()` & `withdrawIncentives()`. Thus, stopping anybody from withdrawing their LP tokens.

The owner of the RocketJoe platform has the ability to enable the emergency withdrawal allowing the depositors to take back their AVAX. But, they lose their burned rJOE tokens and the gas fees.

The dev team might use this attack vector if they think the price of their token is too low. In that case, they can DOS attack the LaunchEvent. If the RocketJoe owner enables the emergency withdrawal, the dev team is able to take back their initial deposit. Thus, they don't lose anything but their reputation.

Proof of Concept

When `createPair()` is called, the function checks whether a pair already exists. If it does, the transaction is reverted: https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L382-L389

Anybody is able to create a new JoePair using the existing TraderJoe contracts. If someone owns both AVAX and the LaunchEvent token, they are able to create a new pair and deposit a small amount of liquidity. Thus, the `totalSupply` will be `> 0`. Meaning, at that point, the call to `createPair()` fails. Per design, the LaunchEvent will be used to issue a token to the public market. So only the dev team and its trusted parties have access to the necessary tokens to create a pair and provide liquidity.

https://github.com/traderjoe-xyz/joe-core/blob/main/contracts/traderjoe/JoeFactory.sol#L30

https://github.com/traderjoe-xyz/joe-core/blob/main/contracts/traderjoe/JoePair.sol#L133

Since `createPair()` can't be executed the `pair` state variable is never initialized: https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L422

Thus, the following two functions are not reachable any more: https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L439

https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L469

If the emergency withdrawal is enabled, the token issuer can take back their deposit: https://github.com/code-423n4/2022-01-trader-

joe/blob/main/contracts/LaunchEvent.sol#L510-L516

Recommended Mitigation Steps

If a LaunchEvent for a token has started, only the LaunchEvent contract should be able to create a JoePair for that token. But, this change has to be made to the contracts that are not in the scope of this audit. I don't think there's a possibility to fix this issue within the RocketJoe contracts.

cryptofish7 (Trader Joe) acknowledged

[M-03] `createRJLaunchEvent()` can be called by anyone with 1 Wei of `_token` and stop others from creating RJLaunchEvent with the same token anymore

Submitted by WatchPug, also found by p4st13r4

https://github.com/code-423n4/2022-01-trader-joe/blob/119e12d715ececc31478e833297f124cc15d27c2/contracts/RocketJoeFactory.sol#L97-L132

```
    function createRJLaunchEvent(
        address _issuer,
        uint256 _phaseOneStartTime,
        address _token,
        uint256 _tokenAmount,
        uint256 _tokenIncentivesPercent,
        uint256 _floorPrice,
        uint256 _maxWithdrawPenalty,
        uint256 _fixedWithdrawPenalty,
        uint256 _maxAllocation,
        uint256 _userTimelock,
        uint256 _issuerTimelock
    ) external override returns (address) {
        require(
            getRJLaunchEvent[_token] == address(0),
            "RJFactory: token has already been issued"
        );
        require(_issuer != address(0), "RJFactory: issuer can't be (
        require(_token != address(0), "RJFactory: token can't be 0 a
```

```
        require(_token != wavax, "RJFactory: token can't be wavax");
        require(
            _tokenAmount > 0,
            "RJFactory: token amount needs to be greater than 0"
        );
        require(
            IJoeFactory(factory).getPair(_token, wavax) == address(0
                IJoePair(IJoeFactory(factory).getPair(_token, wavax)
                    .totalSupply() ==
                0,
            "RJFactory: liquid pair already exists"
        );

        address launchEvent = Clones.clone(eventImplementation);

        // msg.sender needs to approve RocketJoeFactory
        IERC20(_token).transferFrom(msg.sender, launchEvent, _tokenA
```

In the current implementation, `RocketJoeFactory.sol#createRJLaunchEvent()` can be called by anyone with at least 1 Wei of `_token`.

This allows a malicious user or attacker to call `createRJLaunchEvent()` with minimal cost and stop others, especially the platform itself or the rightful issuer of the token from creating the RJLaunchEvent.

Recommendation

Consider making `createRJLaunchEvent()` only callable by the owner of `RocketJoeFactory`.

cryptofish7 (Trader Joe) acknowledged and commented:

> That's the spirit, not a single token should be in circulation.

[M-04] Uninitialized `RocketJoeStaking.lastRewardTimestamp` can inflate `rJoe` supply

Submitted by cmichel

The `RocketJoeStaking.lastRewardTimestamp` is initialized to zero. Usually, this does not matter as `updatePool` is called before the first deposit and when `joeSupply = joe.balanceOf(address(this)) == 0`, it is set to the current time.

```
    function updatePool() public {
        if (block.timestamp <= lastRewardTimestamp) {
            return;
        }
        uint256 joeSupply = joe.balanceOf(address(this));

        // @audit lastRewardTimestamp is not initialized. can send 1
        if (joeSupply == 0) {
            lastRewardTimestamp = block.timestamp;
            return;
        }
        uint256 multiplier = block.timestamp - lastRewardTimestamp;
        uint256 rJoeReward = multiplier * rJoePerSec;
        accRJoePerShare =
            accRJoePerShare +
            (rJoeReward * PRECISION) /
            joeSupply;
        lastRewardTimestamp = block.timestamp;

        rJoe.mint(address(this), rJoeReward);
    }
```

However, if a user first directly transfers `Joe` tokens to the contract before the first `updatePool` call, the `block.timestamp - lastRewardTimestamp = block.timestamp` will be a large timestamp value and lots of `rJoe` will be minted (but not distributed to users). Even though they are not distributed to the users, inflating the `rJoe` total supply might not be desired.

Recommendation

Consider tracking the actual total deposits in a storage variable and using this value instead of the current balance for `joeSupply`. This way, transferring tokens to the contract has no influence and depositing through `deposit` first calls `updatePool` and initializes `lastRewardTimestamp`.

cryptofish7 (Trader Joe) confirmed and commented:

> Fix: https://github.com/traderjoe-xyz/rocket-joe/pull/76

[M-05] Failed transfer with low level call could be overlooked

Submitted by harleythedog, also found by sirhashalot

In `LaunchEvent.sol`, the function `_safeTransferAVAX` is as follows:

```
    function _safeTransferAVAX(address _to, uint256 _value) internal
        (bool success, ) = _to.call{value: _value}(new bytes(0));
        require(success, "LaunchEvent: avax transfer failed");
    }
```

This function is utilized in a few different places in the contract. According to the Solidity docs), "The low-level functions `call`, `delegatecall` and `staticcall` return `true` as their first return value if the account called is non-existent, as part of the design of the EVM. Account existence must be checked prior to calling if needed".

As a result, it is possible that this call will fail, but `_safeTransferAVAX` will not notice anything went wrong. In particular, it is possible that the address `rocketJoeFactory.penaltyCollector()` is a deleted contract (perhaps a security flaw was found and `selfdestruct` was called so that users know to use an updated smart contract), but `_safeTransferAVAX` will not revert. If `rocketJoeFactory.penaltyCollector()` is indeed a non-existent contract, it would be better for `_safeTransferAVAX` to revert until an admin can manually correct the `penaltyCollector` in the factory.

For reference, see a similar high severity reported in a Uniswap audit here (report # 9): https://github.com/Uniswap/v3-core/blob/main/audits/tob/audit.pdf

Proof of Concept

See `_safeTransferAVAX` here. See how this function is called with `_to` as `rocketJoeFactory.penaltyCollector()` here, but this contract's existence is not verified, which is a problem as described above.

Recommended Mitigation Steps

Check for contract existence on low-level calls, so that failures are not missed.

cryptofish7 (Trader Joe) acknowledged

[M-06] possibility of minting rJOE tokens before ownership is changed to RocketJoeStaking

Submitted by hubble

There is a possibility of the rJOE tokens in RocketJoeToken.sol to be minted by original owner without staking any JOE, before the ownership is transferred to RocketJoeStaking

Proof of Concept

Contract : RocketJoeToken.sol Line : 37 function mint(address _to, uint256 _amount) external onlyOwner { _mint(_to, _amount); }

Recommended Mitigation Steps

The transferOwnership(address) function inherited from Ownable.sol is used to change to a new owner i.e., RocketJoeStaking. In the RocketJoeToken.sol contract, define and override this function with an additional check that the totalSupply <= 0

cryptofish7 (Trader Joe) acknowledged

[M-07] withdrawAVAX() function has call to sender without reentrancy protection

Submitted by jayjonah8

In LauchEvent.sol the withdrawAVAX() function makes an external call to the msg.sender by way of _safeTransferAVAX. This allows the caller to reenter this and other functions in this and other protocol files. To prevent reentrancy and cross function reentrancy there should be reentrancy guard modifiers placed on the

withdrawAVAX() function and any other function that makes external calls to the caller.

Proof of Concept

https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L368

https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L370

Recommended Mitigation Steps

Add reentrancy guard modifier to withdrawAVAX() function.

cryptofish7 (Trader Joe) confirmed and commented:

> Resolved using CEI: https://github.com/traderjoe-xyz/rocket-joe/commit/dbd19cc400abb5863edfc0443dd408ba5ae3e99a

[M-08] LP Tokens May Be Locked in Contract Due to `allowEmergencyWithdraw()` in Stage 3

Submitted by kirk-baird

The function allowEmergencyWithdraw() may be called by the `rocketJoeFactory.owner()` at any time. If it is called while the protocol is in Stage 3 and a pair has been created then the LP tokens will be locked and both issues and depositors will be unable to withdraw.

Proof of Concept

If `allowEmergencyWithdraw()` is called `stopped` is set to `true`. As a result functions withdrawIncentives() and withdrawLiquidity() will revert due to the `isStopped(false)` modifier reverting.

Additionally, emergencyWithdraw() will revert since all the `WAVAX` and `token` balances have been transferred to the liquidity pool.

Thus, depositors and issuers will have no methods of removing their LP tokens or incentives.

Recommended Mitigation Steps

Consider adding the requirement `require(address(pair) != address(0), "LaunchEvent: pair not created");` to the function `allowEmergencyWithdraw()`.

cryptofish7 (Trader Joe) confirmed and commented:

> To fix, we allow withdrawal of LP in `emergencyWithdraw()`: https://github.com/traderjoe-xyz/rocket-joe/commit/8a93c43e9972a2cf7c8ee04ccf263a405ecfcecc

[M-09] `createPair()` expects zero slippage

Submitted by sirhashalot

The LaunchEvent.sol `createPair()` function calls router.addLiquidity() with a amountADesired == amountAMin and amountBDesired == amountBMin. Because there is no allowance for slippage, if the zero slippage requirement is not met then the addLiquidity() function will revert and prevent users from using the createPair() function. This could be caused either by frontrunning the createPair call or in a situation where the liquidity pool exists but does not allow for zero slippage with the assets it is holding.

Proof of Concept

The zero slippage addLiquidity call is found in LaunchEvent.sol. This code may have been written with the assumption that only Rocket Joe will have a balance of the new token, so no other user could call the addLiquidity function with both assets, since the whitepaper states “Rocket Joe liquidity launch will complete before launchpad public sale release any tokens to the public”. However, the new token contract should be considered untrusted and Rocket Joe cannot guarantee where all the new tokens are before phase 3 of the Rocket Joe launch event, which is when `createPair()` is called. The token creator who has control over the token

allocation is not controlled by Trader Joe, so an attacker who has early access to the new token can break the outlined assumptions.

Recommended Mitigation Steps

Consider how the launch event functions may break if the new token is launched by an attacker who doesn't follow the assumptions outlined. One solution for this `createPair()` issue is to add an input parameter to the function to handle a slippage allowance.

cryptofish7 (Trader Joe) acknowledged

[M-10] Use safeTransfer/safeTransferFrom consistently instead of transfer/transferFrom

Submitted by cccz, also found by 0x1f8b, bobi, byterocket, Dravee, hack3r-0m, sirhashalot, TomFrenchBlockchain, UncleGrandpa925, and WatchPug

It is good to add a require() statement that checks the return value of token transfers or to use something like OpenZeppelin's safeTransfer/safeTransferFrom unless one is sure the given token reverts in case of a failure. Failure to do so will cause silent failures of transfers and affect token accounting in contract.

Proof of Concept

https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L457

https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L463

https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L489

https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L513

https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/LaunchEvent.sol#L537

Recommended Mitigation Steps

Consider using safeTransfer/safeTransferFrom or require() consistently.

cryptofish7 (Trader Joe) confirmed and commented:

> Fix: https://github.com/traderjoe-xyz/rocket-joe/commit/9e11786ffbfe71f324bc67411270900ec21355fc

LSDan (judge) increased severity from Low to Medium and commented:

> This could result in a loss of funds given the right external conditions.

> `2 — Med (M): vulns have a risk of 2 and are considered "Medium" severity when assets are not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

[M-11] Re-enterable Code When Making a Deposit to Stake

Submitted by kirk-baird, also found by 0v3rf10w and static

Note: this attack requires `rJoe` to relinquish control during `tranfer()` which under the current RocketJoeToken it does not. Thus this vulnerability is raised as medium rather than high. Although it's not exploitable currently, it is a highly risky code pattern that should be avoided.

This vulnerability would allow the entire rJoe balance to be drained from the contract.

Proof of Concept

The function deposit() would be vulnerable to reentrancy if rJoe relinquished control flow.

The following lines show the reward calculations in variable `pending`. These calculations use two state variables `user.amount` and `user.rewardDebt`. Each of these are updated after `_safeRJoeTransfer()`.

Thus if an attacker was able to get control flow during the `rJoe::tranfer()` function they would be able to reenter `deposit()` and the value calculated for `pending` would be the same as the previous iteration hence they would again be transferred `pending` rJoe tokens. During the rJoe transfer the would again gain control of the execution and call `deposit()` again. The process could be repeated until the entire rJoe balance of the contract has been transferred to the attacker.

```
    if (user.amount > 0) {
        uint256 pending = (user.amount * accRJoePerShare) /
            PRECISION -
            user.rewardDebt;
        _safeRJoeTransfer(msg.sender, pending);
    }
    user.amount = user.amount + _amount;
    user.rewardDebt = (user.amount * accRJoePerShare) / PRECISION;
```

Recommended Mitigation Steps

There are two possible mitigations. First is to use the [openzeppelin reentrancy guard] over the `deposit()` function which will prevent multiple deposits being made simultaneously.

The second mitigation is to follow the [checks-effects-interactions] pattern. This would involve updating all state variables before making any external calls.

[cryptofish7 (Trader Joe) confirmed, but disagreed with severity and commented]:

> Disagree with severity

> Fix: https://github.com/traderjoe-xyz/rocket-joe/pull/142

[LSDan (judge) commented]:

> I agree with the warden's assessment of risk on this one. Leaving it unaddressed would represent a potential future compromise if it was forgotten about by the team.

[M-12] Pair creation can be denied

Submitted by cmichel, also found by harleythedog, UncleGrandpa925, and WatchPug

The `LaunchEvent.createPair` requires that no previous pool was created for the `WAVAX <> _token` pair.

```
    function createPair() external isStopped(false) atPhase(Phase.Ph
        (address wavaxAddress, address tokenAddress) = (
            address(WAVAX),
            address(token)
        );
        // @audit grief: anyone can create pair
        require(
            factory.getPair(wavaxAddress, tokenAddress) == address(0
            "LaunchEvent: pair already created"
        );

        // ...
    }
```

A griefer can create a pool for the `WAVAX <> _token` pair by calling `JoeFactory.createPair(WAVAX, _token)` while the launch event phase 1 or 2 is running. No liquidity can then be provided and an emergency state must be triggered for users and the issuer to be able to withdraw again.

Recommendation

It must be assumed that the pool is already created and even initialized as pool creation and liquidity provisioning is permissionless. Special attention must be paid if the pool is already initialized with liquidity at a different price than the launch event price.

It would be enough to have a standard min. LP return "slippage" check (using parameter values for `amountAMin/amountBMin` instead of the hardcoded ones in `router.addLiquidity`) in `LaunchEvent.createPair()`. The function must then be callable with special privileges only, for example, by the issuer. Alternatively, the slippage check can be hardcoded as a percentage of the raised amounts (`amountADesired = 0.95 * wavaxReserve, amountBDesired = 0.95 * tokenAllocated`).

This will prevent attacks that try to provide LP at a bad pool price as the transaction will revert when receiving less than the slippage parameter. If the pool is already initialized, it should just get arbitraged to the auction token price and liquidity can then be provided at the expected rate again.

cryptofish7 (Trader Joe) confirmed, but disagreed with High severity and commented:

> Fix: https://github.com/traderjoe-xyz/rocket-joe/pull/81

> Should be 2 (Medium).

LSDan (judge) decreased severity to Medium and commented:

> This issue would not put assets at risk. but would impact the availability of the protocol for certain pairs.

> ```
> 2 — Med (M): vulns have a risk of 2 and are considered "Medium" severity when assets are not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.
> ```

[M-13] ERC20 return values not checked

Submitted by cmichel, also found by Czar102, defsec, hack3r-0m, hyh, Ruhum, saian, Tomio, and WatchPug

The `ERC20.transfer()` and `ERC20.transferFrom()` functions return a boolean value indicating success. This parameter needs to be checked for success. Some tokens do **not** revert if the transfer failed but return `false` instead. Tokens that don't actually perform the transfer and return `false` are still counted as a correct transfer.

Recommendation

As the Launch event token can be any token, all interactions with it should follow correct EIP20 checks. We recommend checking the `success` boolean of all `.transfer` and `.transferFrom` calls for the unknown `token` contract.

- `LaunchEvent.withdrawLiquidity: token.transfer(msg.sender, amount);`
- `LaunchEvent.withdrawIncentives: token.transfer(msg.sender, amount);`
- `LaunchEvent.emergencyWithdraw: token.transfer(msg.sender, amount);`
- `LaunchEvent.skim: token.transfer(msg.sender, amount);`
- `RocketJoeFactory.createRJLaunchEvent: IERC20(_token).transferFrom(msg.sender, launchEvent, _tokenAmount);`

cryptofish7 (Trader Joe) confirmed and commented:

> Fix: https://github.com/traderjoe-xyz/rocket-joe/commit/dbd19cc4

LSDan (judge) commented:

> Given external factors, this could result in a loss of funds.

> `2 — Med (M): vulns have a risk of 2 and are considered "Medium" severity when assets are not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

[M-14] Incompatibility With Rebasing/Deflationary/Inflationary tokens

Submitted by defsec

The TraderJOE protocol do not appear to support rebasing/deflationary/inflationary tokens whose balance changes during transfers or over time. The necessary checks include at least verifying the amount of tokens transferred to contracts before and after the actual transfer to infer any fees/interest.

Proof of Concept

https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/RocketJoeStaking.sol#L133

https://github.com/code-423n4/2022-01-trader-joe/blob/main/contracts/RocketJoeFactory.sol#L132

Recommended Mitigation Steps

- Ensure that to check previous balance/after balance equals to amount for any rebasing/inflation/deflation
- Add support in contracts for such tokens before accepting user-supplied tokens
- Consider supporting deflationary / rebasing / etc tokens by extra checking the balances before/after or strictly inform your users not to use such tokens if they don't want to lose them.

cryptofish7 (Trader Joe) disputed and commented:

> It won't revert as long as token's balance doesn't decrease (this never happens).

LSDan (judge) increased severity from Low to Medium and commented:

> It is possible for someone to unknowingly use this functionality with a token that rebases down during the launch event. Just because you don't support a token type, doesn't mean that the design doesn't exist. This is a medium risk, not a low risk, because there is the potential for external interaction to cause a loss of funds.

[M-15] Lack of input checks (withrawal penalties should always be greater than 0)

Submitted by pedroais

If penalties are set to 0 the protocol would be vulnerable to price manipulations like the one described in the contest documentation.

Proof of Concept

The protocol uses economic penalties to punish withdraws to protect against economic price manipulation attacks. If these penalties are set to 0 in the creation of a token launch the sale would be vulnerable to this kind of attack. The penalties should never be 0 for any token sale.

The economic attack that could be done with 0 penalties is detailed on page 7 of the whitepaper.

https://github.com/traderjoe-xyz/research/blob/main/RocketJoe_Launch_Platform_for_Bootstrapping_Protocol-Owned_Liquidity.pdf

I consider this to be a medium risk since it could completely invalidate a token launch but it's still unlikely (but possible) the creators will set penalties to 0. This could be done by mistake or by the creators of the launch event to exploit it themselves.

Recommended Mitigation Steps

Require penalties to be greater than 0 either in the initializer function or in the factory.

cryptofish7 (Trader Joe) acknowledged, but disagreed with Medium severity and commented:

> Disagree with severity, should be 1 (Low).

LSDan (judge) commented:

> I agree with the warden on risk here.

Low Risk Findings (26)

- [L-01] Lack of ownership check *Submitted by 0x1f8b*
- [L-02] Missing divide by 0 check on tokenAllocated *Submitted by Dravee*
- [L-03] Inclusive checks in LaunchEvent.sol for time-management *Submitted by Dravee*
- [L-04] Missing inheritances *Submitted by Dravee*
- [L-05] RocketJoeStaking.initialize arguments need to be checked *Submitted by hyh, also found by Funen and wuwe1*
- [L-06] `createRJLaunchEvent()` Multiple `launchEvent` can be created unexpectedly by reentrancy *Submitted by WatchPug, also found by pauliax*
- [L-07] `rJoeAmount` can never be less than the `_avaxAmount` *Submitted by cmichel*
- [L-08] Initialization Function Is Missing If Token is Equals To WAVAX On the LaunchEvent *Submitted by defsec*
- [L-09] Unsafe call to decimals() *Submitted by pauliax*
- [L-10] Not verified function inputs of public / external functions *Submitted by robee*
- [L-11] safeApprove of openZeppelin is deprecated *Submitted by robee*
- [L-12] Mult instead div in compares *Submitted by robee*
- [L-13] Penalty Collector must be trusted *Submitted by cmichel*
- [L-14] FRONT-RUNNABLE INITIALIZERS *Submitted by cccz, also found by Czar102, defsec, jayjonah8, Jujic, kirk-baird, MetaOxNull, p4st13r4, pauliax, robee, Ruhum, and wuwe1*
- [L-15] No Transfer Ownership Pattern *Submitted by cccz, also found by defsec*
- [L-16] Must approve 0 first *Submitted by cccz, also found by csanuragjain, defsec, and robee*
- [L-17] rJoe rewards can be manipulated for all users *Submitted by jayjonah8*
- [L-18] `LaunchEvent.sol` : Use `SafeERC20.safeApprove` in `createPair()` *Submitted by Dravee, also found by 0x1f8b, bobi, defsec, pauliax, and*

WatchPug

- [L-19] Missing consistent zero address checks *Submitted by sirhashalot, also found by 0v3rf10w, 0x1f8b, cccz, defsec, Dravee, gzeon, and UncleGrandpa925*
- [L-20] Incorecct calculation between actual code and comment *Submitted by Funen*
- [L-21] RocketJoeFactory assume the input address is WAVAX *Submitted by cccz*
- [L-22] `LaunchEvent.tokenIncentivesPercent` wrong docs *Submitted by cmichel*
- [L-23] Misleading comment in `LaunchEvent.getReserves` *Submitted by cmichel*
- [L-24] Wrong comment *Submitted by wuwe1*
- [L-25] Admin Deny of Service *Submitted by 0x1f8b*
- [L-26] LaunchEvent pays out fewer incentives then expected *Submitted by TomFrenchBlockchain*

Non-Critical Findings (6)

- [N-01] Code Style: non-constant should not be named in all caps *Submitted by WatchPug*
- [N-02] Missing commenting *Submitted by robee*
- [N-03] Reasonable upper limits for phase durations *Submitted by pauliax*
- [N-04] Missing event emitting *Submitted by wuwe1, also found by cmichel, defsec, and pedroais*
- [N-05] Unused variable _amount *Submitted by p4st13r4*
- [N-06] The staking contract should have pause/unpause functionality. *Submitted by defsec*

Gas Optimizations (68)

- [G-01] `RocketJoeStaking.sol#withdraw` has an unneeded require statement *Submitted by 0x0x0x*
- [G-02] "> 0" is less efficient than "!= 0" for unsigned integers *Submitted by WatchPug, also found by 0x0x0x, byterocket, Czar102, defsec, Dravee, gzeon,*

Jujic, MetaOxNull, pedroais, robee, Ruhum, and solgryn

- [G-03] Gas Optimization: Variables that could be set immutable *Submitted by gzeon, also found by 0x1f8b, bobi, csanuragjain, Ruhum, and WatchPug*
- [G-04] Use short reason strings can save gas *Submitted by WatchPug, also found by byterocket, Czar102, defsec, Dravee, Jujic, MetaOxNull, p4st13r4, pauliax, robee, and sirhashalot*
- [G-05] Use Shift Right/Left instead of Division/Multiplication if possible *Submitted by Czar102, also found by byterocket, d4rk, and Dravee*
- [G-06] Gas Optimisation - Simplify `_atPhase()` Logic *Submitted by kirk-baird, also found by Czar102, and pauliax*
- [G-07] Error never thrown *Submitted by Czar102*
- [G-08] Use constructors *Submitted by Czar102*
- [G-09] Gas Optimisation - Reduce storage loads *Submitted by kirk-baird, also found by Czar102*
- [G-10] Gas: Mark functions as payable when users can't mistakenly send ETH *Submitted by Dravee*
- [G-11] Adding unchecked directive can save gas *Submitted by WatchPug, also found by defsec, Dravee, hyh, Jujic, Rhynorater, TomFrenchBlockchain, and ye0lde*
- [G-12] Gas: Tight variable packing in `LaunchEvent.sol` *Submitted by Dravee*
- [G-13] Functions can be external *Submitted by sirhashalot, also found by Dravee, Rhynorater, and robee*
- [G-14] Gas: Missing checks for non-zero transfer value calls *Submitted by Dravee*
- [G-15] Gas: Non-strict inequalities are cheaper than strict ones *Submitted by Dravee*
- [G-16] Gas in `LaunchEvent.sol:createPair()` : calculation should get cached *Submitted by Dravee*
- [G-17] Gas in `LaunchEvent.sol:withdrawLiquidity()` : `address(pair)` should get cached *Submitted by Dravee*
- [G-18] Gas in `LaunchEvent.sol:emergencyWithdraw()` : `user.balance` should get cached earlier *Submitted by Dravee*

- [G-19] Gas in `RocketJoeFactory.sol:_emitLaunchedEvent()` : a value used only once shouldn't get cached *Submitted by Dravee*
- [G-20] Gas in `RocketJoeStaking.sol:deposit()` : `user.amount` should get cached and used for calculation *Submitted by Dravee*
- [G-21] Gas in `RocketJoeStaking.sol:withdraw()` : `user.amount` should get cached and used for calculation *Submitted by Dravee*
- [G-22] Gas in `RocketJoeStaking.sol:withdraw()` : `accRJoePerShare` should get cached *Submitted by Dravee*
- [G-23] Gas in `RocketJoeStaking.sol:deposit()` : `accRJoePerShare` should get cached *Submitted by Dravee*
- [G-24] Gas in `RocketJoeStaking.sol:updatePool()` : `lastRewardTimestamp` should get cached *Submitted by Dravee*
- [G-25] Gas in `RocketJoeFactory.sol:createRJLaunchEvent()` : `wavax` should get cached *Submitted by Dravee*
- [G-26] Gas in `LaunchEvent.sol:currentPhase()` : `auctionStart` and `PHASE_ONE_DURATION` should get cached *Submitted by Dravee*
- [G-27] Gas in `LaunchEvent.sol:createPair()` : `wavaxReserve` should get cached *Submitted by Dravee*
- [G-28] Gas in `LaunchEvent.sol:withdrawLiquidity()` : `tokenReserve` should get cached earlier *Submitted by Dravee*
- [G-29] Gas in `LaunchEvent.sol:emergencyWithdraw()` : `issuer` should get cached *Submitted by Dravee*
- [G-30] Gas in `LaunchEvent.sol:getPenalty()` : `PHASE_ONE_DURATION` and `PHASE_ONE_NO_FEE_DURATION` should get cached *Submitted by Dravee*
- [G-31] Gas in `LaunchEvent.sol:pairBalance()` : `wavaxAllocated` should get cached *Submitted by Dravee*
- [G-32] Saving more gas by using `immutable phase` *Submitted by Funen*
- [G-33] Caching `rJoe` variable *Submitted by Jujic*
- [G-34] Mint() by OnlyOwner Lack of Zero Address Check for Address _to *Submitted by MetaOxNull*

- [G-35] Cache external call results can save gas *Submitted by WatchPug, also found by byterocket, hyh, kirk-baird, Ruhum, TomFrenchBlockchain, and WatchPug*
- [G-36] Cache and read storage variables from the stack can save gas *Submitted by WatchPug, also found by robee, Ruhum, and TomFrenchBlockchain*
- [G-37] Storing phase durations rather than start times duplicates calculations *Submitted by TomFrenchBlockchain, also found by pauliax*
- [G-38] Timestamps/durations held in storage can be packed *Submitted by TomFrenchBlockchain*
- [G-39] maxWithdrawPenalty and fixedWithdrawPenalty can be packed together *Submitted by TomFrenchBlockchain*
- [G-40] Free gas savings for using solidity 0.8.10+ *Submitted by TomFrenchBlockchain*
- [G-41] Use clones with immutable variables to reduce costs from SLOADs *Submitted by TomFrenchBlockchain*
- [G-42] Explicit initialisation variable wastes gas. *Submitted by TomFrenchBlockchain*
- [G-43] UserData struct can be packed into a single slot. *Submitted by TomFrenchBlockchain*
- [G-44] using `unchecked` can save gas *Submitted by Tomio*
- [G-45] `RocketJoeFactory.sol#createRJLaunchEvent()` Check of `_issuer != address(0)`, `_token != address(0)`, `_tokenAmount > 0` can be done earlier to save gas *Submitted by WatchPug*
- [G-46] Gas: `RocketJoeStaking.withdraw` *Submitted by cmichel*
- [G-47] Gas savings *Submitted by csanuragjain*
- [G-48] Gas Optimization: fmul optimization *Submitted by gzeon*
- [G-49] Gas Optimization: Use type(uint256).max instead of block.timestamp *Submitted by gzeon*
- [G-50] Gas Optimziation: Unnecessary pairBalance call *Submitted by gzeon*
- [G-51] Gas Optimisation - Unnecessary External Calls in `LaunchEvent.initialize()` *Submitted by kirk-baird*

- [G-52] Separate issuer functions from regular users *Submitted by pauliax*
- [G-53] Unchecked math operations *Submitted by pauliax*
- [G-54] Unchecked math operations *Submitted by pauliax*
- [G-55] Repeated storage access *Submitted by pauliax*
- [G-56] Cheaper operation should be done first in an if statement *Submitted by pedroais*
- [G-57] instead of using && in require. just use require multiple time *Submitted by rfa*
- [G-58] using += to save gas *Submitted by rfa*
- [G-59] Check if amount is not zero to save gas *Submitted by robee*
- [G-60] Cache powers of 10 used several times *Submitted by robee*
- [G-61] Internal functions to private *Submitted by robee*
- [G-62] Mark unchanging variables immutable *Submitted by Czar102*
- [G-63] Missing Sanity Checks Will Cause To Revert On the Function *Submitted by defsec, also found by Tomio*
- [G-64] The contracts use unlocked pragma *Submitted by hyh, also found by bobi, byterocket, Czar102, defsec, Dravee, jayjonah8, Jujic, and p4st13r4*
- [G-65] Redundant type casting *Submitted by WatchPug, also found by 0x1f8b*
- [G-66] `Ownable` library is redundant *Submitted by WatchPug, also found by Dravee, TomFrenchBlockchain, and wuwe1*
- [G-67] Check if amount > 0 before token transfer can save gas *Submitted by WatchPug, also found by jayjonah8*
- [G-68] Useless storage variable *Submitted by p4st13r4, also found by 0x1f8b*

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct

formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top