Protofire VE8020-Launchpad Security Analysis Report and Formal Verification Properties

December 2023

Table of Contents

Summary

This document describes the specification and verification of the ve8020-Launchpad using the Certora Prover and manual code review findings. The work was undertaken from **27th November 2023** to **15th December 2023**. The latest commit that was reviewed is [d1e79a2].

The following contracts list is included in the **scope**:

- `RewardDistributor.sol`
- `RewardFaucet.sol`
- `VotingEscrow.vy`
- `Launchpad.vy`

The contracts were verified against the Solidity versions:

- RewardDistributor.sol : solc 7.6
- RewardFaucet: solc 8.18
- OpenZeppelin ERC20 instances: solc 8.18

The Certora Prover demonstrated that the implementation of the Solidity contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all Solidity contracts. During the verification process and the manual audit, the Certora Prover discovered bugs in the Solidity contracts code, as listed below.

Summary of findings

The table below summarizes the issues discovered during the audit, categorized by severity.

Severity	Total discovered	Total fixed	Total acknowledged
High	2	2	2
Medium	1	0	1
Low	8	4	8
Informational	0		0
Total (High, Medium, Low)	**11**	**6**	**11**

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

Main Issues Discovered

Issue-01: Sum of weekly rewards isn't conserved in RewardFaucet

Severity: High
Probability: High
Category: Insolvency
Property violated: "For any token, the sum of token week rewards from all weeks is equal to total rewards for that token".
File(s): RewardFaucet.sol
Bug description: movePastRewards() doesn't preserve the sum of token week amounts, since the week to which the tokens are moved is updated with a wrong value.
Exploit scenario:
When one calls movePastRewards(), the tokenWeekAmounts[address][uint256] mapping value is nullified for the past week, and updated for the week to which the rewards are moved. The update is erroneous, since it simply overrides the mapping value instead of increasing it with the correct value.
Implications: The sum of rewards over all weeks is not preserved, hence some rewards would be lost, if moved.
Protofire's response: Acknowledged and fixed in commit d1e79a2.

Issue-02: Admin can frontrun and cause loss of funds

Severity: High

Probability: Low

Category: Front-run by admin (centralization)

File(s): VotingEscrow.vy

Bug description: The admin can frontrun a call of a user to "withdraw_early" and change the K penalty factor, thus causing unexpected loss of funds for the user.

Exploit scenario: The K penalty factor is set to 0 (as determined by the admin). A user wants to withdraw his money, but his funds are still locked. Therefore, he calls the function "withdraw_early". Since the K factor is 0, no loss of funds are expected (by incurring a penalty for leaving early). However, the admin frontruns the user's call to withdraw early and sets the K factor to 50. Consequently, the user will lose a significant amount of money unexpectedly.

.

Example:

Max_time = 1 year

Time_Left = 1/5 year

Locked_funds = 1000

K = 0

Expected behavior:

Penalty = 1000 * 0 ((1/5 year) / 1 year) / 10 = 0

Amount left = 1000 – 0 = 1000

Behavior after frontrun by the admin:

Changed by the admin - K = 50

Penalty = 1000 * 50 * ((1/5 year) / 1 year) / 10 = 1000

Amount left = 1000 – 1000 = 0

Protofire's response: Acknowledged and fixed in commit d1e79a2.

Issue-03: Contract initialization is not guarded

Severity: Medium
Probability: Low
Category: Access control
Property violated: "Only the admin can change the admin".
File(s): RewardDistributor.sol
Bug description: The initialize() function that sets the contract admin and start time isn't protected thus could be called by anyone, who can set the admin to his will.
Exploit scenario: Calling to initialize() in the RewardDistributor contract isn't protected by any access modifier, thus could potentially be called by anyone, if not called right after deployment. This is not expected to happen in the case where the contract is deployed using the launchpad (Launchpad.vy), where the initialization is done together with deployment. If an independent user chooses to initialize separately, anyone could front-run and "steal" the ownership.
Protofire's response: Acknowledged. Not related to project (project flow).

Issue-04: User claimed tokens calculation can overflow

Severity: Low
Probability: Low
Category: Arithmetic overflow
Property violated: No overflows\underflows are possible.
File(s): RewardDistributor.sol
Bug description: The calculation of the claimed amount for a user could overflow, reducing extremely the amount that is rewarded to a user.
Exploit scenario: An extreme case where a user's voting power is close to max uint128 and the tokens per week exceed max uint128 could lead to the claim tokens calculation for that user to overflow, if the product of these two numbers exceed max uint256. The contract is compiled in a Solidity version < 0.8, which allows overflow and underflow of arithmetic calculations. Suppose a user has a really large voting power (~2^127) as a result of locking a high decimal token in the VE, and also, the tokens per week in the RewardDistributor could aggregate to a number beyond 2^128. This seems impossible at first since the contract could not have more than 2^128 - 1 tokens at any time, but if one deposits and claims consecutively several times, the balance of the contract will stay bounded as intended, but the tokens per week only increases, which could eventually cross the intended bound of max uint128. In such an extreme case (if one has deposited a significant amount of tokens into the contract), the amount to be claimed calculated in `_claimToken()` will overflow for that powerful user. Note that anyone could call the claim operation on behalf of anyone else so access control could not block this scenario from happening.
Protofire's response: Acknowledged and fixed in 5fd2ef5 .

Issue-05: Loss of voting power due to rounding

Severity: Low

Probability: High

Category: Loss of rewards funds

File(s): VotingEscrow.vy

Bug description: The calculation of the voting power per user involves dividing the amount of locked tokens by the MAX_TIME that could lead to rounding-down of the voting power to a lower amount, and sometimes zero.

.

Exploit scenario:

Calling “create_lock” with 30 USDC:

MAX_TIME = 1 year (31,104,000 seconds)

Locked amount = 30 USDC (30,000,000 due to precision of USDC)

Slope calculated = 30,000,000 / 31,104,000 = 0

Bias calculated = slope * Time_Left = 0 (doesn’t matter what is Time_Left)

Protofire’s response: Acknowledged: “There is no damage (and risk) to user funds. And also the possible loss of rewards is quite low and only likely for small quantities and some tokens”.

Issue-06: Reward distributor start time isn’t bounded

Severity: Low
Probability: Low
Category: Unbounded value
Property violated: “Any time cursor is bound by max uint64”.
File(s): RewardDistributor.sol
Bug description: The start time of the distributor contract isn’t bounded by any reasonable value upon initialization, which can lead to the contract being bricked.
Exploit scenario: Upon calling to initialize() in the RewardDistributor contract, the msg.sender sets the start time of the contract, the time point from which all actions are approved, including deposit and checkpointing. A check for a reasonable value for this parameter is missing, so an unintentional error by the user could lead to the contract being bricked.
Protofire’s response: Acknowledged and fixed in commit d1e79a2.

Issue-07: 10th past week isn’t distributed

Severity: Low

Probability: HIgh

Category: Denial Of Service

File(s): RewardFaucet.sol

Bug description: Past 10th week is not available for distribution.

.

“distributePastRewards” function is applicable for distributing rewards from the current week and 9 prior weeks.

as can be seen on the loop on line 157:

```
for (uint256 i = 0; i < 10; ++i) {

            uint256 amount = tokenWeekAmounts[token][weekStart];

            if (amount == 0) {

                weekStart -= 1 weeks;

                continue;

            }

            tokenWeekAmounts[token][weekStart] = 0;

            totalAmount += amount;

            weekStart -= 1 weeks;

        }
```

In order to withdraw rewards from prior weeks, a user can call the “movePastRewards” function.

This function is applicable for distributing rewards starting only from past 11 weeks or more,

as can be seen on the require on line 194:

```
uint256 pastWeekStart = _roundDownTimestamp(pastWeekTimestamp);

        require(pastWeekStart < _roundDownTimestamp(block.timestamp) - 10 weeks,
'!outdate');
```

There is a 1 week gap, at the 10th past week, between the functions that enable withdraw of past weeks rewards,

which makes the rewards distribution for this week unaccessible.

Protofire’s response: Acknowledged and fixed in commit d1e79a2.

Issue-08: A user can call “withdraw_early” without paying any penalty due to rounding

Severity: Low

Probability: Low

.

Category: Fee/penalty avoidance

File(s): VotingEscrow.vy

Bug description:

The formula for calculating the penalty is:

(Locked_amount *(Time_left * 10^18 / MAX_TIME* K)/10^18 / 10

That could lead to a round down of the penalty to a lower amount.

Exploit scenario:

Calling “withdraw_early”:

k = 1

Locked amount = 30 USDC (30,000,000 due to precision of USDC)

Time_Left = 5 hours (18,000 seconds)

MAX_TIME = 100,000 year (3,144,960,000,000 seconds)

Penalty ratio = Time_Left * 10^18 / MAX_TIME * K = 18000 * 10^18 / 3,144,960,000,000 * 1 = 5,707,762,557

Penalty = penalty_ratio * Locked_amount / 10^18 / 10 = 0

Protofire’s response: Acknowledged.

Issue-09: Checkpointing is enabled for disallowed tokens

Severity: Low
Probability: Low
Category: Undesired behavior
Property violated: “There are no rewards for disallowed tokens.” , “The contract cannot increase\decrease the balance of its disallowed tokens.”.
File(s): RewardDistributor.sol
Bug description: A user can checkpoint a disallowed token and update the rewards amounts for that token, if he has sent some tokens to the contract before.
Exploit scenario: While one cannot deposit or claim disallowed tokens using the contract interface, one can bypass this restriction and send these tokens directly to the contract, followed by checkpointing this token, which is allowed. While the user most probably won’t drain any funds from the protocol, it would be better to block any functionality for this token, if it is activated in the future.

Protofire's response: Acknowledged and fixed in commit d1e79a2.

Issue-10: claimToken() might reward less tokens than expected

Severity: Low
Probability: Low
Category: Denial of Service
Property violated: "The later in time a user claims tokens, the more his token cursor advances in time."
File(s): RewardDistributor.sol
Bug description: If one calls `claimToken()` exactly in the beginning of the week that wasn't yet checkpointed, the `_checkpointTotalSupply()` call doesn't update the time cursor and the first unclaimable week of the user would become earlier than expected, leading to accumulation of tokens from a smaller number of weeks.
Exploit scenario: Suppose one week wasn't yet checkpointed, such that the global time cursor still points to one of the past weeks. Then, one user might want to claim his tokens by calling the `claimToken()` function which, before rewarding him his tokens, will call `_checkpointTotalSupply()` first. If the time stamp at which this transaction is executed is exactly equal to the beginning of the week timestamp, then the condition at the beginning of the function (line 714 at _checkpointTotalSupply()) will make the function exit early without advancing the time cursor. The implication of this "mistake" is that once the flow enters the `_claimToken()` function, the first unclaimable week of the user could be earlier than expected, hence the user would withdraw less tokens, for less weeks in total. This coincidental case could only prevent the user from claiming his tokens at the very transaction, but would not prevent him from doing so at a later transaction, in a different block.

Protofire's response: Acknowledged.

Issue-11: The tokens distributed per week can miss a few tokens per checkpoint

Severity: Low
Probability: High
Category: Loss of rewards
File(s): RewardDistributor.sol
Bug description: Upon a call to `checkpointToken()` (internal or external), the contract checks the total amount of new tokens that were added to it since the last checkpoint or claim (with respect to the cached balance), and distributes

that delta among the last weeks. Rounding error could cause losing one wei in each week distributed.

Exploit scenario: The call to `_checkpointToken()` checks the difference between the last cached balance of that token to the current ERC20 balance of the distributor contract and distributes that delta among the last weeks that weren't distributed. If more than one week should gain rewards, the delta is distributed proportionally for each week according to the (partial) week length with respect to the last time since the checkpoint. The sum of distributed tokens isn't necessarily the same original delta because of rounding-down, thus each week can lose 1 wei for every iteration. Since the cached delta is updated to the current balance of the contract, that lost dust of funds could not be re-distributed and are lost (inside the contract).

Protofire's response: Acknowledged.

Formal Verification

The structure of properties:

1. <notation> <property description>

Function names (and signatures) shall be written in Source code Pro font size 11 with the gray highlight, e.g., `foo(uint256)`

Notations

✅Indicates the rule is formally verified.
❌Indicates the rule is violated.
⏳Indicates the rule is timing out.

VotingEscrow.vy assumptions

The behavior of the VotingEscrow, which is used in the verification of the reward Solidity contracts, was replaced by a generic mock. The mock is supposed to replace the actual functions with abstract functions that return arbitrary values, restricted to some assumptions.
The following conditions were assumed about the VotingEscrow.vy, that were checked manually:

- The maximal epoch for each user is less than 2^64.
- The maximal valid time stamp for any history point is less than 2^64 - 10 weeks (in seconds).
- The time stamp (Point.ts) for a user history is monotonically increasing with the user epoch.
- The zero address latest epoch is 0. Thus it has no registered voting power.
- The voting escrow total supply at any week timestamp is equal or larger than any user ve-balance at the same week.

RewardFaucet.sol properties

Assumptions

- Loop unrolling: We assume any loop can have at most 3 iterations.

Properties

1. ✅ One cannot initialize the contract twice.
2. ✅ Calling initialize sets the 'initialized' flag to true.

3. ❌ For any token, the sum of tokens week rewards from all weeks is equal to total rewards for that token. Issue-01: Sum of weekly rewards isn't conserved in RewardFaucet
   ✅ Verified after fix commit.

4. ✅ The `depositExactWeek()` action is amount-additive.
5. ✅ `depositEqualWeeksPeriod()` adds an equal amount of rewards per each week (except for the last and first weeks).
6. ✅ `depositEqualWeeksPeriod()` transfers to the distributor an amount that is equal to the total tokens to be rewarded divided by weekCount.
7. ❌Any past week tokens can be distributed. Issue-07: 10th past week isn't distributed.
   ✅ Verified after fix commit.
8. ✅ Contract's ERC20 token balance is always greater or equal to the total token rewards.
9. ✅ Sum of past week and next week amounts before and after calling `movePastRewards()` is preserved.
10. ✅ Contract's ERC20 token balance and Contract's rewards balance before and after calling `movePastRewards()` are preserved.
11. ✅ No action can DOS a call to `distributePastRewards()`.[1]
12. ✅ No action can DOS a call to `depositExactWeek()`.
13. ✅ No action can DOS a call to `movePastRewards()`.

RewardDistributor.sol properties

Assumptions

- Loop unrolling: We assume any loop can have at most 3 iterations. This means loops always reach a break() statement at some point.
- The maximal valid time stamp for any block less than 2^64 - 10 weeks (in seconds).
- Some rules assume the contract is already initialized. If the contract is not initialized right after deployment (as expected from the Launchpad interface), then unexpected behavior could happen when checkpointing tokens or voting power.
- The voting escrow total supply at any week timestamp is equal or larger to any user ve-balance at the same week.

Properties

1. ✅ It's impossible to initialize the contract twice.
2. ✅ The token is allowed if and only if it's in the allowed token array.

[1] When the action is depositEqualWeeksPeriod, we limit the check to weekCount = 1.

3. ✅ If the contract is initialized, then it's admin is non-zero.
4. ✅The initialize function sets the admin address as expected.
5. ❌Only the admin can change the admin. Issue-03: Contract initialization is not guarded
6. ❌ There are no rewards for disallowed tokens. Issue-09: Checkpointing is enabled for disallowed tokens
   ✅ Verified after fix commit.
7. ❌ The contract cannot increase\decrease the balance of its disallowed tokens. Issue-09: Checkpointing is enabled for disallowed tokens
   ✅ Verified after fix commit.
8. ✅ If the contract is initialized, its start time is always in the past.
9. ✅The user token time cursor is measured in weeks.
10. ✅ The tokens per week are always zero for weeks after the token cursor.
11. ✅ The tokens per week are always zero for weeks after the token cursor.
12. ✅ The global time cursor, the start time and the cursor of any user always correspond to initial weeks.
13. ✅The user start time is never later than the user cursor.
14. ✅The last checkpointed epoch is bounded by the escrow user epoch.
15. ✅The token start time is never later than the token cursor.
16. ❌Any time cursor is bounded by max uint64. Issue-06: Reward distributor start time isn't bounded
    ✅ Verified after fix commit.
17. ✅The time cursor is never in the future (or equal to start time).
18. ✅The token time cursor is never in the future.
19. ✅The user-token time cursor is never in the future.
20. ✅No other actor can make some user balance decrease.
21. ✅The user VE balance at a timestamp after or equal to its cursor must be zero.
22. ✅The zero address time cursor is zero.
23. ✅ No other actor can cause any user balance to decrease.
24. ✅ The user balance at a timestamp after or equal to its cursor must be zero.
25. ✅ It's impossible to update any user ve balance more than once.

26. ❌ It's impossible to update the distributed tokens per week more than once.
27. ✅Checkpointing again in the same block has no effect.
28. ✅Checkpointing token again in the same block has no effect[2].
29. ✅Checkpointing user again in the same block has no effect.
30. ✅ Checkpointing again in the same block shouldn't revert.
31. ✅ Checkpointing token again in the same block shouldn't revert.
32. ✅ Checkpointing user again in the same block shouldn't revert.
33. ✅ Checkpointing a token doesn't affect the balance and cursor of another.
34. ✅ Checkpointing a user doesn't affect the balance and cursor of another.
35. ✅ Checkpointing a token cannot DOS the checkpoint of another.
36. ✅ Checkpointing a user cannot DOS the checkpoint of another.
37. ✅ Checkpointing tokens is commutative with respect to tokens order.
38. ✅Checkpointing users is commutative with respect to users order.
39. ✅ No action can DOS the `checkpoint()` action.
40. ✅ No action can DOS the `checkpointToken()` action.
41. ✅ No different actor can prevent someone from claiming his tokens. One cannot DOS a call to `claimToken()` [3].
42. ✅ The tokensPerWeek() can only change for one token at a time.
43. ✅ After claiming non-zero tokens, the cursor must advance in time.
44. ✅ The user balance cannot change for a timestamp behind the user cursor.
45. ❌The later in time a user claims tokens, the more his token cursor advances in time. Issue-10: claimToken() might reward less tokens than expected
46. ✅ Any time cursor of any kind cannot decrease.
47. ✅ The veSupply is changed (or accessed) for whole week timestamps.
48. ✅ Cannot claim tokens for week before the user cursor.

[2] The only effect of an immediate token checkpoint is increasing the tokensPerWeek[token][currentWeek] by an amount that doesn't exceed the total available tokens to distribute before the first checkpoint.
[3] Assuming the protocol is always solvent with respect to the claimed token. Which means that the contract token balance is large enough to cover all claim calls. If solvency isn't assumed, then this property could be violated.