$SCALE Buy & Burn

Security review

Version 1.0

Reviewed by
nmirchev8
deth

Table of Contents

1 About Egis Security

Egis Security is a team of experienced smart contract researchers, who strive to provide the best smart contract security services possible to DeFi protocols.

The team has a proven track record on public auditing platforms like Code4rena, Sherlock, and Cantina, earning top placements and rewards exceeding $170,000. They have identified over 150 high and medium-severity vulnerabilities in both public contests and private audits.

2 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	Scale Buy & Burn Contract
Repository	https://github.com/e-losiev/scale-bb-contract
Commit hash	798463da9a55646bc852de1e175b9b95d0f7428a
Resolution	9a578daba3a6439518bc488890f52e6f7a474492
Methods	Manual review, Unit testing, Fuzz testing, Invariant testing

Test suite can be found under "test/foundry/" directory

Scope

contracts/ScaleBuyBurn.sol

Issues Found

Critical risk	0
High risk	0
Medium risk	1
Low risk	2
Informational	3

5 Findings

5.1 Medium risk

5.1.1 If bnb is not `whitelistedTo`, for Element280, `_swapDragonXforELMNT` may return wrong value resulting in revert

Severity: *Medium risk*

Context: ScaleBuyBurn.sol#L186-L191

Description:

Element280 token is a fee-on-transfer token, which can have whitelisted addresses. If current buy and burn contract is not `whitelistedTo`, `_swapDragonXforELMNT` return value (update e280 balance of contract) will be wrong, because it is calculated without the fee:

```
        address[] memory path = new address[](2);
        path[0] = DRAGONX;
        path[1] = E280;

        uint256[] memory amounts = IUniswapV2Router02(UNISWAP_V2_ROUTER).
            swapExactTokensForTokens(
            amountIn, minAmountOut, path, address(this), deadline
        );

        return amounts[1];
```

You can notice that we return `amounts[1]`, which is a raw amount calculated from uniswapV2 `UniswapV2Library.getAmountsOut(factory, amountIn, path)`, which is the amount being transferred out from `uniswapPool`, but it is not guaranteed that our contract receives the amount. This issues can result in updating `e280Balance` to an amount larger than the balance in reality, which will later revert in `_swapELMNT`:

```
        uint256 swappedAmount = _swapDragonXforELMNT(amountToSwap, minE280Amount,
            deadline); // @sus are we sure we recieve the whole ‘swappedAmount‘ here
            ?
        unchecked {
            return currentE280Balance + swappedAmount;
        }
```

PoC can be found here under the `testBuyAndBurnWithDragonXSwapFailsWhenBalanceOf280IsBelowCapAfterSwap` test.

Recommendation: Inside `_swapDragonXforELMNT` introduce the following snippet to reflect the actual balance update:

```
        uint256 balanceBefore = E280.balanceOf(address(this));
        uint256[] memory amounts = IUniswapV2Router02(UNISWAP_V2_ROUTER).
            swapExactTokensForTokens(
            amountIn, minAmountOut, path, address(this), deadline
        );

        return E280.balanceOf(address(this)) - balanceBefore;
```

Resolution: Acknowledged

5.2 Low risk

5.2.1 E280 being swapped is always < `capPerSwapE280`

Severity: *Low risk*

Context: ScaleBuyBurn.sol#L64-L65

Description: `capPerSwapE280` param is used to limit the amount being swapped at once to avoid slippage. But we first limit the amount of e280 to be max `capPerSwapE280`, then use this amount to transfer the incentive fee to the caller and then we swap the updated balance, which will always be `capPerSwapE280 - incentiveFee`.

Recommendation: Consider if that's the intended design.

Resolution: Acknowledged

5.2.2 Consider emitting events on important state changes

Severity: *Low risk*

Context: Migrator.sol#L19

Description: Consider emitting events on the following state changes, in case you want to index and process state changes over time: - `setIncentiveFee` - `setBuyBurnInterval` - `setCapPerSwapE280` - `setCapPerSwapDragonX` - `setWhitelisted`

Resolution: Acknowledged

5.3 Informational

5.3.1 Cache `capPerSwapE280` in buyAndBurn to safe gas

Severity: *Informational*

Context: ScaleBuyBurn.sol#L64-L65

Description: Consider caching `capPerSwapE280` in `buyAndBurn` contract to safe gas instead of fetching it each time from the storage.

Resolution: Fixed

5.3.2 Change `getBuyBurnParams` function logic

Severity: *Informational*

Context: ScaleBuyBurn.sol#L128-L139

Description: In `getBuyBurnParams` consider setting `dragonXAmount` only when `additionalSwap = true`.

Resolution: Fixed