·Decurity·

Smart Contract Security Audit Report

Dyson Finance

Decurity, 2022

Contents

1. General Information

This report contains information about the results of the security audit of the Dyson Finance (hereafter referred to as “Customer”) smart contracts, conducted by Decurity in the period from 09/27/2022 to 10/10/2022.

1.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

1.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/Gabriel-Dyson/dyson-audit-2022-09-26. Initial review was done for the commit 533839d232403d701ef6ae5d77cb1ba3347ce6fc.

1.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

1.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2. Summary

As a result of this work, we have discovered high-risk security issues that could lead to the financial impact.

The other suggestions included fixing the low-risk issues and some best practices (see 3.1).

The Dyson team has given the feedback for the suggested changes and explanation for the underlying code.

2.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of Oct 10, 2022.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Possibility of arbitrage inside one pool	DysonPair.sol	High	Acknowledged
Funds may be locked due to insufficient liquidity	DysonPair.sol	High	Acknowledged
Anyone can withdraw tokens and ether from DysonRouter	DysonRouter.sol	Medium	Acknowledged
Unlimited allowance on migration	sDYSON.sol	Medium	Not Fixed
Fees are not checked for transfers	Bribe.sol, DysonPair.sol, DysonRouter.sol	Medium	Not Fixed
Anyone can withdraw tokens from DysonPair that were deposited to 0x0	DysonPair.sol	Low	Not Fixed

address			
No time lock when changing the farm parameters	Farm.sol	Low	Not Fixed
Bypassing the withdrawal delay	Gauge.sol	Low	Not Fixed
Possibility of the constant product decrease	DysonPair.sol	Low	Not Fixed
DoS via staking of sDYSON tokens for other accounts	sDYSON.sol	Low	Not Fixed
ERC20 approval race condition	Dyson.sol, sDYSON.sol	Low	Not Fixed
Multiple missing 0x0 check on input addresses	Agency.sol, Dyson.sol, DysonFactory.sol, sDYSON.sol, DysonPair.sol, DysonRouter.sol, Farm.sol, Gauge.sol	Low	Not Fixed
Appropriate events are not emitted	Dyson.sol, Farm.sol, Gauge.sol, sDYSON.sol	Low	Not Fixed
Function should be declared external	Dyson.sol	Info	Not Fixed
Unused state variable	DysonRouter.sol	Info	Not Fixed
Notes on gauge pools and bribing	Gauge.sol, Bribe.sol	Info	Not Fixed

3. General Recommendations

This section contains general recommendations on how to fix discovered weaknesses and vulnerabilities and how to improve overall security level.

Section 3.1 contains a list of general mitigations against the discovered weaknesses, technical recommendations for each finding can be found in section 4.

Section 3.2 describes a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level.

3.1. Current Findings Remediation

Follow the recommendations in section 4.

3.2. Security Process Improvement

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

4. Findings

4.1. Possibility of arbitrage inside one pool

Risk Level: High

Contracts:

- DysonPair.sol

References:

- https://docs.uniswap.org/protocol/V2/guides/smart-contract-integration/providing-liquidity

Description:

The extraction of premiums from the pool balance and choosing only one single side to add liquidity is changing the composition ratio of assets in the pool which makes it vulnerable to arbitrage.

Proofs:

Consider the following test case:

```
//uint immutable INITIAL_LIQUIDITY_TOKEN0 = 10**30;
//uint immutable INITIAL_LIQUIDITY_TOKEN1 = 10**30;

function testDepositProfitDeposit() public {
    uint depositAmount = 10**30;
    uint swapAmount = 10 * 10**11;
    uint output0;
    uint output1;

    vm.startPrank(bob);
    pair.deposit1(bob, depositAmount, 0, 1 days);
```

```
        skip(20 minutes);
        output0 = pair.swap0in(bob, swapAmount, 0);
        skip(1 days);
        pair.withdraw(0);
        pair.deposit0(bob, depositAmount, 0, 1 days);
        skip(20 minutes);
        output1 = pair.swap1in(bob, output0, 0);
        skip(1 days);
        pair.withdraw(1);
        assertTrue(swapAmount < output1);
    }
```

As a result of swaps, we were able to withdraw from the pair two and a half times as many tokens as was sent in the beginning.

This test assumes that no other transactions happen between the deposits and swaps. However, in reality other arbitrageurs may come and execute transactions in between. To eliminate this, an attacker may use MEV-like methods to exclude the undesired transactions from the blocks.

Remediation:

Do not pay premiums from the pool reserves.

4.2. Funds may be locked due to insufficient liquidity

Risk Level: High

Contracts:

- DysonPair.sol

Description:

If in a pair there is low liquidity and a user wants to deposit an amount that is significantly larger than the amount of liquidity in the pool, then he or she will be unable to withdraw their funds from the pool. Consider the following parameters:

- token0 initial liquidity - 1 million tokens (18 decimals)
- token1 initial liquidity - 1 million tokens (18 decimals)
- deposit period - 1 day
- deposit amount - 69 million tokens

In such a situation after waiting for a lockup period (1 day), if there are no other deposits in the pool, a user will be unable to withdraw their funds as the pool will try to return more tokens than it actually has.

Proofs:

Consider the following test case:

```
        function testWithdrawNoLiquidity(uint depositAmount) public
{
            vm.assume(depositAmount > 0);
            vm.startPrank(bob);
            pair.deposit1(bob, depositAmount, 0, 1 days);
            skip(1 days);
            pair.withdraw(0);
        }
```

It will fail when deposit amount is 68232047051938575323141529 since the pair does not have enough liquidity:

```
Failing tests:
Encountered 1 failing test in src/test/DysonPair.t.sol:DysonPairTest
[FAIL. Reason: TransferHelper: TRANSFER_FAILED Counterexample: calldata=0x94c13e680000000000000000000000
0000000000000000000003870b2aeadfb482ca29599, args=[68232047051938575323141529]] testWithdraw1(uint256)
(runs: 43, μ: 157461, ~: 156614)
```

Remediation:

Consider pre-calculating the amount of withdrawn tokens for the specified period and checking that the pool has enough liquidity before letting users deposit amounts that are significantly larger than the current reserves.

4.3. Anyone can withdraw tokens and ether from DysonRouter

Risk Level: Medium

Contracts:

- DysonRouter.sol

References:

- https://swcregistry.io/docs/SWC-105

Description:

The functions `withdrawETH` and `swap0ETHOut` expect an argument `pair` which can be an arbitrary contract instead of `DysonPair`. Thus, a malicious user can craft a contract which follows `IDysonPair` interface, but contains malicious logic that will allow them to withdraw any ERC20 token or ether that DysonRouter holds. Basically, this issue is equivalent to `rescueERC20` not having an `onlyOwner` modifier and additionally unwrapped ether can be withdrawn via `swap0ETHOut`.

Proofs:

Below is the test code that demonstrates the attack:

```
    function testNormalPairSwap01Hacked() public {
            address pair = address(normalPair); // pair of
(Dyson,Dyson) tokens
        uint swapAmount = 10**18;
        uint output;
```

```
        uint AliceBalance;
        uint newAliceBalance;

        vm.startPrank(bob);
        IWETH(WETH).deposit{value: 1 ether}();
        IWETH(WETH).transfer(address(router), 1 ether);
        assertEq(IWETH(WETH).balanceOf(address(router)), 1 ether);

        changePrank(alice);
        AliceBalance = alice.balance;
        output = router.swap0ETHOut(pair, alice, swapAmount, 0);
        newAliceBalance = alice.balance;
        assertEq(output, newAliceBalance - AliceBalance);
        assertEq(IWETH(WETH).balanceOf(address(router)), 0 ether);
    }
```

Remediation:

Pair address should not be supplied externally by the user. Instead, consider passing token addresses of the pair. Pair addresses should be retrieved from the `getPair` mapping of the DysonFactory contract.

4.4. Unlimited allowance on migration

Risk Level: Medium

Contracts:

- sDYSON.sol

References:

- https://kalis.me/unlimited-erc20-allowances/

Description:

The function `migrate` in sDYSON.sol performs an unlimited approval of sDYSON tokens (line 286) to the new staking contract:

```
_approve(msg.sender, migration, type(uint).max);
```

It may be a bad pattern since it enables an attacker to withdraw users' funds in case there is a flaw that lets them spend the tokens on behalf of the new staking contract.

Remediation:

Consider allowing a new staking contract to spend only the amount of sDYSON tokens that a user actually has:

```
_approve(msg.sender, migration, vault.sDYSONAmount);
```

4.5. Fees are not checked for transfers

Risk Level: Medium

Contracts:

- Bribe.sol
- DysonPair.sol
- DysonRouter.sol

References:

- https://github.com/compound-finance/compound-protocol/blob/6548ec3e8c733d18cd4bdb2c21197f666f828f35/contracts/CErc20.sol#L156

Description:

The "input" or "amount" from parameters should not be used as a trusted value in functions. These parameters may not reflect the actual value being transferred (e.g. when there is a fee on transfer).

Remediation:

Consider checking the actual amount received by contract after the transfer function has been executed.

4.6. No timelock when changing the farm parameters

Risk Level: Low

Contracts:

- Farm.sol

Description:

No timelock that controls changes in the farm parameters such as “weight” and “rewardRate” via the `setGlobalRewardRate` function.

Remediation:

Consider adding a timelock to control the parameters of the pool that can be set by the farm.

4.7. Bypassing the withdrawal delay

Risk Level: Low

Contracts:

- Gauge.sol

Description:

In the case of a deposit to the Gauge contract at the end of the week, the user does not need to wait a week to withdraw his funds.

For example, if the user called the `applyWithdrawl` function when “block.timestamp” is equal to 1665619199 then “_week” would be equal to 2753. After that user can withdraw after 100 seconds when the block.timestamp would be equal to

1665619299, because 1665619299/604800 = 2754 which means that there is a new week started.

Remediation:

Consider changing the check from the number of the week to the amount of time.

4.8. Anyone can withdraw tokens from DysonPair that were deposited to 0x0 address

Risk Level: Low

Contracts:

- DysonPair.sol

Description:

Tokens that were deposited to 0x0 address can be withdrawn by anyone via the `withdrawWithSig` function. In functions deposit0() / deposit1() there is no check that the argument `to` cannot be zero. Additionally, `withdrawWithSig` is using `_ecrecover` which returns 0x0 address in case it gets the wrong input parameters. Thus, a malicious user just needs to call the `withdrawWithSig` function with an incorrect signature to retrieve all the tokens that were deposited to zero address.

Remediation:

Consider checking that an input address `to` and return value of `_ecrecover` are not zero.

4.9. DoS via staking of sDYSON tokens for other accounts

Risk Level: Low

Contracts:

- sDYSON.sol

Description:

`dysonAmountStaked` and `votingPower` functions are using loops to iterate over users' vaults. It's possible to block execution of these functions for any user via adding a large number of the vaults using the `stake` function which leads to exceeding gas hard limit.

Remediation:

Remove the possibility to stake for any address.

4.10. ERC20 approval race condition

Risk Level: Low

Contracts:

- Dyson.sol
- sDYSON.sol

References:

- https://www.adrianhetman.com/unboxing-erc20-approve-issues/
- https://swcregistry.io/docs/SWC-114

Description:

In the standard ERC20 implementation approve() overrides current allowance and it doesn't increase/decrease allowance. If a token holder wants to decrease a previously set allowance, he or she might be front-run by the spender to spend the previous allowance before the new `approve` call comes through. It is possible due to mempool transparency and gas price manipulation which may help to get a transaction executed before another one.

Remediation:

Consider implementing increaseAllowance() / decreaseAllowance() functions in `Dyson` and `sDYSON` ERC20 contracts which do not overwrite the allowance but dynamically change it depending on the current value. Refer to OpenZeppelin's implementation:

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol#L181

4.11. Multiple missing 0x0 check on input addresses

Risk Level: Low

References:

- https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Description:

There are multiple occurrences of dangerous assignment of the input address to a storage variable without proper sanitization, namely not checking that the address is zero. Such practice may lead to unavailability of the contract if a critical storage variable such as `owner` is set to zero. There are the following occurrences:

- Variable _owner in Agency.constructor in Agency.sol (line 70)
- Variable _owner in DYSON.transferOwnership in Dyson.sol (line 62)
- Variable _owner in DYSON.constructor in Dyson.sol (line 31)
- Variable _controller in DysonFactory.setController in DysonFactory.sol (line 42)
- Variable _controller in DysonFactory.constructor in DysonFactory.sol (line 19)
- Variable _feeTo in DysonPair.setFeeTo in DysonPair.sol (line 298)
- Variable _WETH in DysonRouter.constructor in DysonRouter.sol (line 26)
- Variable _owner in DysonRouter.constructor in DysonRouter.sol (line 27)
- Variable _owner in Farm.constructor in Farm.sol (line 60)
- Variable _owner in Farm.transferOwnership in Farm.sol (line 71)
- Variable _sgov in Gauge.constructor in Gauge.sol (line 67)
- Variable _owner in Gauge.transferOwnership in Gauge.sol (line 82)
- Variable _migration in sDYSON.setMigration in sDYSON.sol (line 204)
- Variable _owner in sDYSON.transferOwnership in sDYSON.sol (line 136)
- Variable dyson in sDYSON.constructor in sDYSON.sol (line 93)

- Variable _owner in sDYSON.constructor in sDYSON.sol (line 92)

Remediation:

Always check that an input address is not zero before assigning a storage variable.

4.12. Appropriate events are not emitted

Risk Level: Info

Contracts:

- Dyson.sol
- Farm.sol
- Gauge.sol
- sDYSON.sol

References:

- https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

Description:

Critical operations regarding access control should always be tracked via events. Particularly, without emitting an appropriate event it is difficult to track the new owner when ownership is transferred. There are the following cases when an event is not emitted:

- Function DYSON.transferOwnership in Dyson.sol (line 62)
- Function Farm.transferOwnership in Farm.sol (line 71)
- Function Gauge.transferOwnership in Gauge.sol (line 82)
- Function sDYSON.transferOwnership in sDYSON.sol (line 136)

Remediation:

Consider emitting an event when ownership is transferred, e.g. `OwnershipTransferred(address previousOwner, address newOwner)` as in an OpenZeppelin library:

https://docs.openzeppelin.com/contracts/2.x/api/ownership#Ownable-OwnershipTransferred-address-address-

4.13. Function should be declared external

Risk Level: Info

Contracts:

- Dyson.sol

References:

- https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

Description:

Function `removeMinter` should be declared as `external` instead of `public` for optimization purposes since it is not called by the contract itself.

Remediation:

Consider setting the visibility of `removeMinter` as external.

4.14. Unused state variable

Risk Level: Info

Contracts:

- DysonRouter.sol

References:

- https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
- https://swcregistry.io/docs/SWC-131

Description:

The state variable `BALANCE_BASE_UNIT` declared on line 20 in DysonRouter.sol is never used.

Remediation:

Consider removing `BALANCE_BASE_UNIT`.

4.15. Notes on gauge pools and bribing

Risk Level: Info

References:

- https://curve.readthedocs.io/dao-vecrv.html

Description:

Dyson documentation uses terms “gauge” and “bribes” apparently referring to Curve’s vote escrowed governance tokens (veCRV). However, Dyson’s implementation radically differs from the original idea.

1) Core principle of vote-escrowed tokens is that they are non-transferrable, however sDYSON token can be transferred but cannot be unwrapped back to DYSON by another person.
2) A user’s balance of vote-escrowed tokens, and hence the voting power, gradually decreases towards the end of the locking period, thus incentivising a longer period of token lockup. Dyson’s voting power is the same along the whole lockup period.
3) A particular gauge weight is directly impacted by the users who vote with their ve-tokens, however in Dyson a gauge weight is set by the admin.

5. Appendix

5.1. About us

The Decurity (former DeFiSecurity.io) team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.