PERIMETER

Tapioca DAO TapToken

Testing Reinforcement Report

06/18/2024

Supervised By:
Rappie | Lead Fuzzing Specialist
rappie@perimetersec.io

0xScourgedev | Lead Fuzzing Specialist
0xscourgedev@perimetersec.io

Prepared By:
nican0r | Junior Fuzzing Specialist
https://x.com/nican0r

Table of Contents

Services Provided 3
Files in Scope 4
Methodology 5
Issues Found
- MED-01: Circular constructor arguments in TapiocaOptionLiquidityProvision and TapiocaOptionBroker 6
- INFO-01: No check for expiry time on aoTAP 8
- INFO-02: timestampToWeek return value isn't as expected 9

Services Provided

Perimeter has successfully delivered a comprehensive suite of services that include:

- **Test Suite Development:**
    - All test contracts in scope were refactored to fix compilation errors due to changes in the contracts being tested.
    - Individual tests were refactored if they resulted in a failing test in order to account for changes to the tested contracts' logic.
    - Certain test contracts were expanded to include stateless fuzzing.
- **Creation of a Final Report:**
    - Created this final report, which includes our methodology, with all findings and their corresponding PoCs, providing a comprehensive overview of the engagement's outcomes.

Files in Scope

The engagement focuses on the files listed below, acquired from commit [f31083e14482c1552728326ac3f2ba5343f12138](f31083e14482c1552728326ac3f2ba5343f12138).

File
test/governance/twTAP.t.sol
test/helpers/TapTestHelper.t.sol
test/helpers/TestUtils.t.sol
test/option-airdrop/AirdropBroker.t.sol
test/option-airdrop/aoTAP.t.sol
test/options/oTAP.t.sol
test/options/TapiocaOptionBroker.t.sol
test/options/TapiocaOptionLiquidityProvision.t.sol
test/tokens/TapToken.t.sol
test/tokens/TapTokenMock.sol
test/tokens/TapTokenMultiCompose.t.sol
test/AirdropBrokerTest.t.sol
test/LTap.t.sol
test/Vesting.t.sol

Files Out of Scope

Files outside the scope were not directly considered in achieving the target. However, since many of these files are utilized by those within the scope, a significant portion was indirectly covered.

Methodology

As the existing test suite had already been created, this engagement focused on adapting the existing tests to work with changes that had been made to the core tap-token repository contracts.

Test contracts were first evaluated for compatibility with the changes made to the contract being tested since the tests had been implemented. Because the majority of test contracts did not compile, they had to be refactored in order to work with the new contract implementations.

After achieving compilation, the majority of the test contracts had failing tests due to changes in the tested contracts' logic which required debugging, with the majority of failing tests being resolved by changing the test contract setup or changing individual tests to handle different values returned by the tested contracts.

Additionally, certain contracts such as the LTap token had significant changes to their contract interface and logic which required a complete refactoring of its tests to work with this new implementation and achieve meaningful coverage.

Stateless fuzzing tests were also added where beneficial in order to provide greater certainty of correct operation in unit tests.

The naming convention for the fuzz tests created includes wrappers, fuzz and implementation annotations:

- Wrappers evaluate a single value for a unit test implementation(suffixed with _wrapper)
- Fuzz tests take a random input value to evaluate the implementation (prefixed with testFuzz)
- Implementations hold the unit test logic and assertions (prefixed with test_, no suffix)

While modifying and running the tests, some issues were uncovered.

MED-01: Circular constructor arguments in TapiocaOptionLiquidityProvision and TapiocaOptionBroker

Severity

Medium

Description

`TapiocaOptionBroker` has a constructor argument for the address of the `TapiocaOptionLiquidityProvision` contract at:

TapiocaOptionBroker.sol#L121
```
tOLP = TapiocaOptionLiquidityProvision(_tOLP);
```

`TapiocaOptionLiquidityProvision` has a constructor argument for the address of the `TapiocaOptionBroker` contract at:

TapiocaOptionLiquidityProvision.sol#L100
```
tapiocaOptionBroker = _tob;
```

Both of the arguments are then set to an immutable variable, thus making it impossible to change post-deployment. As one contract is required in the other's constructor argument, one of the contracts must be deployed before the other with the wrong address, which can not be changed.

Proof of concept

Both contracts also have functions with these addresses, and will cause unexpected behavior, below are some examples:

TapiocaOptionBroker.sol#L186
```
tOLPLockPosition = tOLP.getLock(oTAPPosition.tOLP);
```

TapiocaOptionBroker.sol#L215
```
(assetId, totalDeposited, weight, isInRescue) =
tOLP.activeSingularities(_singularity);
```

TapiocaOptionLiquidityProvision.sol#L246
```
if (tokenOwner == tapiocaOptionBroker) revert TobIsHolder();
```

Impact

This causes the initial deployment to have incorrect values, and they are not able to be changed due to the immutability of the variables. Thus it would require a change in the contracts and a re-deployment to have the correct values.

Recommendation

Implement a restricted function that allows setting the address of one of these variables to allow for the addresses to be set correctly.

Response

INFO-01: No check for expiry time on aoTAP

Severity

Informational

Description

`aoTAP::mint` is missing a check on the expiry time passed in:

aoTAP.sol#L102-118

```
 function mint(address _to, uint128 _expiry, uint128 _discount, uint256
_amount, uint64 _phase)
        external
        nonReentrant
        returns (uint256 tokenId)
    {
        if (msg.sender != broker) revert OnlyBroker();

        tokenId = ++mintedAOTAP;
        AirdropTapOption storage option = options[tokenId];
        option.expiry = _expiry;
        option.discount = _discount;
        option.amount = _amount;
        option.phase = _phase;

        _safeMint(_to, tokenId);
        emit Mint(_to, tokenId, option);
    }
```

This allows a user to mint an aoTAP token with an expiry time that has passed.

Proof of concept

See `test_mint_past_expiry_time` for an example where a user is able to mint an aoTAP token with an expiry time that has already passed.

Recommendation

Include a check in `aoTAOP::mint` that checks if the value for expiry passed in has already passed.

Response

INFO-02: timestampToWeek return value isn't as expected

Severity

Informational

Description

In the `TapiocaOptionBroker::test_timestamp` test the return value of timestampToWeek in the originally defined test for return_value3 is expected to be 0 if passing in a value less than the length of an epoch but if `TapiocaOptionBroker::emissionsStartTime` is unset it returns 1.

TapiocaOptionBroker.sol#L160-L167

```
function timestampToWeek(uint256 timestamp) external view returns (uint256) {
        if (timestamp == 0) {
            timestamp = block.timestamp;
        }
        if (timestamp < emissionsStartTime) return 0;

        return _timestampToWeek(timestamp);
    }
```

Proof of concept

See `TapiocaOptionBrokerTest::test_timestamp` failing case.

Recommendation

If this is not intended behavior modify timestampToWeek to include a check for if emissionsStartTime is set and return 0 if not.

Response