CD SECURITY

AUDIT REPORT

TradFiLines

October 2023

Introduction

A time-boxed security review of **TradFiLines** protocol was done by **CDSecurity**, with a focus on the security aspects of the application's implementation.

The protocol was audited by:

- ddimitrov22
- chrisdior.eth
- Stoicov
- Todorov

Disclaimer

A security review of a smart contract can never assure the absolute absence of vulnerabilities. It's a bounded effort that depends on available time, resources, and expertise to uncover as many vulnerabilities as possible. There's no guarantee of achieving 100% security after the review, nor is it guaranteed to identify any issues with your smart contracts. As a best practice, it's strongly recommended to follow up with additional security reviews, bug bounty programs, and on-chain monitoring for continued vigilance.

About TradFiLines

TradFiLines is a protocol of 500 distinct TFL NFT's on the Ethereum blockchain that hold claims to the same number of ordinals on the Bitcoin blockchain. Previously wrapped NFTs through `TFLColour` can now be wrapped to receive an animation. Each user is incetivized to wrap its NFT with a small amount of ETH. The NFTs can only be traded within the regular trading hours of the NYSE/NASDAQ exchanges(9:30 to 16:00 EST). A user can call the `bribe` function and pay the `bribeAmount` to extend the trading window by 1 hour. The `BokkyPooBahsDateTimeLibrary` is implemented to get the specific dates and to determine wether it is a daylight savings period.

The newest functionality added to the protocol, the `CircuitBreaker`, makes it possible to halt trading hours for a given day in the event that the price has moved above certain threshold. An interesting design choice is to use the BAYC Chainlink oracle to get the floor price on which is based if the trading can be halted. This can be done by calling the `tripBreaker` function which can be called by anyone and it is incentivized upon a call making it work trustless and autonomously.

Privileged Roles and Actors

- `owner` - can withdraw contracts' `ETH` balances, swap the `oracle` and `tradFiRenderer` addresses, change the amounts of fees and incentives and toggle incentives on and off.
- `users` - The TradFiLines NFT holders which can wrap and trade them and halt the trading for specific days.
- `TradFiLinesRenderer` - the renderer is responsible for generating the length and amount of bars for the NFTs. However, once `locked` it can't be changed anymore.

Severity classification

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

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

Scope

The scope for this security review was the following:

- `CircuitBreaker.sol`
- `MockOracle.sol`
- `TFLAUtil.sol`
- `TrafFiLinesAnimation.sol`
- `TradFiLinesRendererAnimation.sol`

The following number of issues were found, categorized by their severity:

- Critical and High : 4 issues
- Medium : 1 issue
- Low : 8 issues
- Informational : 3 issues

Findings Summary

ID	Title	Severity
[H-01]	`CircuitBreaker` can not incentivize users	High
[H-02]	The `setValue()` will revert because of a time unit	High
[H-03]	An off-by-one error can DoS the `bribe` method	High
[H-04]	Using an oracle for another collection can DoS the trading	High
[M-01]	Rewards for wrapping NFTs can be gamed	Medium
[L-01]	A user can bribe an hour in the past and be charged for it	Low

ID	Title	Severity
[L-02]	Unchecked low level call return value	Low
[L-03]	Use a two-step ownership transfer process	Low
[L-04]	Discrepancies between the code and the docs	Low
[L-05]	Use call() instead of transfer() when sending ETH	Low
[L-06]	Using an old version of OpenZeppelin is dangerous	Low
[L-07]	The value `idToCategory` mapping will always be the default one	Low
[L-08]	An event is emitted with default value `address(0)`	Low
[I-01]	Variables can be turned into an `immutable`	Informational
[I-02]	Use custom errors instead of require statements	Informational
[I-03]	Redundant code	Informational

Detailed Findings

[H-01] `CircuitBreaker` can not incentivize users

Severity

Impact High, as users will be discouraged to call an important function for the protocol

Likelihood High, as there is no way in which `CircuitBreaker` can receive `ETH`

Vulnerability Details

It is stated in the docs that calls to the methods `tripBreaker` and `setValue` are incentivized in order to make them work trustlessly and autonomously. However `CircuitBreaker.sol` has no `receive()` , no `fallback()` or `payable` function which means that there is no way `ETH` to be sent to the contract. Because of this, whoever calls `tripBreaker` or `setValue` won't receive the expected reward which will lead to discouraging users to call the `tripBreaker` which is the main functionality of the contract. Thus, instead of not receiving reward, the users will be at loss because of the gas fee. In addition a `withdraw` method is present in the contract which shows that the initial intention for `CircuitBreaker` is to be able to receive `ETH` yet this is not possible.

Recommendations

Consider enabling `CircuitBreaker` to receive `ETH` if you want to use the incentive mechanism mentioned in the docs.

Client - fixed

[H-02] The `setValue()` will revert because of a wrong time unit

Severity

Impact High, as the function won't work

Likelihood High, as the function will revert every time for more than a year

Vulnerability Details

The function `setValue` in `CircuitBreaker.sol` allows you to update the oracle floor value by reading the Chainlink oracle.

Now lets take a look at the second require statement in it:

```
require(updatedAt - lastTimestamp > 1000*60*60*12, "The value can only be updated after 12 hours. If the price moves a lot before 12 hours have passed, the breaker should be called");
```

The math `1000*60*60*12` is equal to 43,200,000 which is expected to be 12 hours like the revert string says. And yes it is 12 hours but in milliseconds and we are comparing it to a unix timestamp values taken from the oracle `updateAt` and `lastTimestamp` which are in seconds. So 43,200,000 seconds will be equal to 500 days and not 12 hours. Which will DOS the function because it will revert every time if the oracle is fetching the right data.

Recommendation

Create a storage variable for example `uint256 public updateTime" = 12 hours;` which is 43,200 seconds. And now you can re-write the require statement as:

```
require(updatedAt - lastTimestamp > updateTime, "The value can only be updated after 12 hours");
```

Client - fixed

[H-03] An off-by-one error can DoS the `bribe` method

Severity

Impact Medium, as the trading window will not be extended by 1 hour

Likelihood High because even if the user pays the `bribeAmount` the function will revert

Vulnerability Details

The purpose of the `bribe` function is to allow the TradFiLines NFT holders to trade them for one extra hour outside of the NYSE/NASDAQ trading hours. This can be done by paying the `bribeAmount` which is set to `0.01 ether` initially. However, a wrong comparison operator in the `require` statement will revert if the `bribeAmount` is equal to `0.01 ether`.

```
    function bribe(uint year, uint month, uint day, uint hour) payable
external {
        require(!bribedHour[year][month * 10000 + day * 100 + hour], "This
hour is already bribed for and open");
        require(msg.value > bribeAmount, string(abi.encodePacked("The
required bribe amount is ", Strings.toString(bribeAmount)))); //@audit -
off by one error
        ...
    }
```

The problem is that when the function reverts, the thrown error will say that exactly the `bribeAmount` is required making it confusing for the user. A user will potentially try again with the same amount of `ether` but the function will revert again because `msg.value` should be more than `0.01 ether`. This will discourage users to use the `bribe` function putting it in a state of DoS.

Recommendation

Change the `require` statement as below:

```diff
-require(msg.value > bribeAmount)
+require(msg.value >= bribeAmount)
```

Client - fixed

[H-04] Using an oracle for another collection can DoS the trading

Impact High because the trading of the NFTs will be halted

Likelihood Medium as this will depend on the floor price of irrelevant collection

Vulnerability Details

The main functionality of the `CircuitBreaker` contract is to halt the trading when the `percentageChange > haltPercentage`. This check is inside the `tripBreaker` function and allows users to disable the trading of the NFTs:

```
function tripBreaker() external {
        int currentValue;
        uint updatedAt;
        (,currentValue,,,updatedAt) = CO.latestRoundData();
        ...
        if(uint(percentageChange) > haltPercentage) {
            haltedDays[date] = true;
         ...
            emit BreakerTripped(uint(percentageChange), nftOracleAddress,
date);
        }
    }
```

The problem arises from the fact that the Chainlink Oracle for BAYC(Bored Ape Yacht Club) is used. The `latestRoundData` call will return the floor price of the BAYC collection which will be used to calculate the `percentageChange`. The two NFT collections are absolutely irrelevant and taking a look at the analytics proves that. For example, the BAYC floor price went from `24.47 ETH` on October 1st to `27.97 ETH` on October 2nd while the floor price of TradFiLines was stable at `0.8 ETH` for the same period.

Another concern is that the floor price for the last 15 days of the TradFiLines is stable at `0.7 ETH` while the floor price of BAYC is quite volatile, bouncing between `24 ETH` and `27+ ETH`.

This could lead to a scenario where the actual `percentageChange` is less than the `haltPercentage` but the movements of the floor price of the BAYC collection will allow users to call `tripBreaker` successfully. This will DoS the protocol for at least a day. It could be even more problematic if the floor price is volatile (as we have seen in the past) and the trading is halted for consecutive days.

Recommendation

As there is no oracle for the TradFiLines collection floor price, there is no easy solution to this issue. Consider removing the function to disable the trading or implement require checks which are not dependent on irrelevant external factors.

Client - acknowledged

[M-01] Rewards for wrapping NFTs can be gamed

Impact Medium because the project will have to pay more ETH in rewards

Likelihood Medium because the NFT holders will likely want to receive higher reward

Vulnerability Details

The `prize` structure for wrapping an NFT is restarted everyday. The first user to wrap his NFT will receive `0.1 ETH`, the second `0.05 ETH`, and all users after that will get `0.015 ETH`. However, as this is restarted each day, users will be incentivized to wait for the next day and get the higher reward. This could lead to users intentionally waiting and not wrapping their NFTs, slowing the whole process(which is expected to take place for 1-2 weeks), or alternatively many users missing the period to wrap their NFTs.

Recommendation

Consider implementing a reward structure that is not on a daily basis.

Client - acknowledged

[L-01] A user can bribe an hour in the past and be charged for it

The `bribe` functionality is meant to extend the trading hours for a given day in exchange for a `bribeAmount` which is currently set to 0.01 `ETH`. However there is no check to prevent bribing for a past period of time. A user can mistakenly pass 2022 as the `year` input followed by the correct `month` and `day` according to the time when he is calling the method. If the hour that is passed as input to the `bribe` method has not been bribed before the result will be that the user will pay the `bribeAmount` but won't be able to take advantage of the functionality leading to bad user experience. Consider checking if the hour which the user wishes to bribe for as a timestamp is not less than `block.timestamp`.

Client - fixed

[L-02] Unchecked low level call return value

There are two occasions of unchecked low level call return values in the protocol. Both of them are observed in `CircuitBreaker.sol`. The first occurrence is in the method `setValue`. This method updates the price and it's incentivized by sending `incentiveUpdatesValue` to the `caller`.Although this functionality should be off by default as the `incentivizedUpdates` `bool` is initially set to false ,the return value of the call to `msg.sender` is not checked which means that the transfer might silently fail(whenever `owner` toggles `incentivizedUpdates` to `true`) and the user will not receive his reward , which on the other hand can lead to bad reputation for the protocol. The second occurrence of this problem is again in `CircuitBreaker.sol` this time in the method `tripBreaker`. The case here is the same , the `caller` might not receive his reward for calling this function because the return value of the call is not checked. Consider adding the `require(sent, "Failed to send Ether");` after the calls in these two methods.

Client - fixed

[L-03] Use a two-step ownership transfer process

Transferring ownership of a contract in a two-step process is always the preferred approach. With this method the new `owner` must accept the ownership role after it has been offered to them. This additional step can help mitigate the risk of unauthorized ownership transfers and accidental changes of ownership. Consider using OpenZeppelin `Ownable2Step` library instead of `Ownable`.

Client - acknowledged

[L-04] Discrepancies between the code and the docs

It is stated in the documentation that trading hours are set in accordance with NYSE/NASDAQ exchanges between the time window 9:30 to 16:00. However in `isWithinOpeningHours()` from `TradFiLinesAnimation.sol` there is the following check:

```
if(hour < 9 || hour > 15) return false;
```

which is a discrepancy between the docs and implementation. A user would think he can trade to 16:00 but in reality trading will finish at 15:00 and the user will have to `bribe` to be able to trade.

Another difference is that the `prize` for wrapping the NFTs. In the documentation is stated that after the second wrapped NFT for the day, the `prize` will be `0.01` for wrapping a NFT while in the code it is hardcoded to `0.015 ether`. Consider aligning the docs and the code.

Client - acknowledged

[L-05] Use `call()` instead of `transfer()` when sending ETH

Couple of functions in the contract are using the `transfer` method to withdraw the `TradFiLinesAnimation.sol`'s balance or wrongly sent ETH in the contract to the `owner`. The `owner`'s address is possible to be a smart contract or a multisig that have a `receive` or `fallback` function that takes up more than the 2300 gas which is the limit of `transfer`. Even if the owner is not a smart contract or a multisig after deployment, the owner may decide to transfer ownership to one and then it might be a problem.

More

Client - fixed

[L-06] Using an old version of OpenZeppelin is dangerous

Currently the protocol is using version `4.7.0` of the OpenZeppelin contracts which are continuously updated to eliminate any bugs and vulnerabilities. Consider using the latest `5.0.0` version.

Client - acknowledged

[L-07] The value `idToCategory` mapping will always be the default one which can cause problems

In `TradFiLinesAnimation.sol` we have a mapping called `idToCategory` which is never updated anywhere in the codebase. Now lets take a look at `tokenURI()`:

```
return
            string(
                abi.encodePacked(
                    tfrm.renderFromSeed(seedsAndInfo,
idToCategory[_tokenId])
                )
            );
```

Here we can see that the value of `idToCategory` mapping is used as a param for `renderFromSeed()` from key `_tokenId`. Here it doesn't matter what is the `_tokenId` because whatever it is, the value of that key will be the default value since the mapping is never updated. Hence `renderFromSeed()` will always get empty string as an input parameter which can be problematic. Either update the mapping or remove it, if it is not important.

Client - fixed

[L-08] An event is emitted with default value `address(0)`

Inside the `CircuitBreaker` contract, there is a variable `address public nftOracleAddress`. The problem is that it is not set anywhere but the `BreakerTripped` event is emitted with the default value which is `address(0)`. This could lead to errors in the off-chain monitoring and logs. Additionally, the input parameter in the constructor is with the same name meaning the variables are shadowed. Consider changing the input variable with an underscore as a prefix - `_nftOracleAddress` to avoid any unexpected behavior.

Client - fixed

[I-01] Variables can be turned into an `immutable`

The `bpbdtc` variable in `CircuitBreaker.sol` is set in the constructor and cannot be changed after meaning it can be declared `immutable`. The same applies for all the variables set in the `constructor` of `TradFiLinesAnimation.sol` except for `tfrm` because as mentioned in the docs, this contract could be changed by the owner under critical circumstances.

Client - fixed

[I-02] Use custom errors instead of require statements

Custom errors are available from solidity version 0.8.4. Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. Consider replacing the require statements with `if (something) revert CustomError()` type of checks.

Client - acknowledged

[I-03] Redundant code

The `owner` of `CircuitBreaker.sol` is initialized twice to the same `owner` (deployer). In `CircuitBreaker's constructor` the method `Ownable._transferOwnership` is called setting the `owner` to `msg.sender`. This is redundant as the `Ownable` constructor itself is setting the `deployer` to be the `owner`. Consider removing the call to `_transferOwnership`.

The `CircuitBreaker` contract is unnecessary importing `Context` as this is inherited by default because of the `Ownable` library. Consider removing the import.

The `bribesActive` boolean in `TradFiLinesAnimation` is not used anywhere and can be removed. The `bribeStatus` in the same contract can be set by calling `setBribeStatus` but is not checked anywhere and its only functionality is to be changed, making both the function and the boolean redundant.

Client - fixed