SOLIDIFIED

Audit Report for BattleFly OTC - April 24, 2023

Summary

Audit Report prepared by Solidified covering the BattleFly OTC smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on April 24, 2023, and the results are presented here.

Audited Files

The source code has been supplied in a compressed ZIP format with the following SHA256: hash: `1156e49035e448ad935157745dea2e7a10e9812d047d3cdf22d04b9492948771`

Update: Fixes were received on Thursday May 18, 2023 and supplied in a private source code repository: https://github.com/BattleFly-Game/battlefly-smart-contracts
Commit hash: `cd29de67b682ba133ed39bf2af84e47189e00a95`

File list:

```
./OTCDiamond
├── OTCInitFacet.sol
├── facets
│   ├── OTCAdminFacet.sol
│   ├── OTCMarketFacet.sol
│   └── OTCVestingFacet.sol
├── helpers
│   ├── DiamondOwnable.sol
│   ├── Errors.sol
│   └── LibDiamond.sol
├── interfaces
│   ├── IDiamondCut.sol
│   ├── IGFly.sol
│   ├── IGFlyStaking.sol
│   ├── IOTCAdmin.sol
│   ├── IOTCMarket.sol
│   ├── IOTCVesting.sol
│   └── IVestedGFly.sol
├── libraries
│   ├── OTCMarketLib.sol
```

```
│   └── OTCVestingLib.sol
├── storage
│   ├── OTCAdminStorage.sol
│   ├── OTCMarketStorage.sol
│   └── OTCVestingStorage.sol
└── utils
    └── Mixins.sol
```

Intended Behavior

BattleFly is a PvP/P2E GameFi project that uses TreasureDAO's MAGIC token as its primary currency.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	-
Level of Documentation	Low	-
Test Coverage	Medium	-

Issues Found

Solidified found that the BattleFly OTC contracts contain 2 critical issues, 2 major issues, 9 minor issues, and 12 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	OTCMarketLib.sol: Arithmetic underflow can lock funds	Critical	Resolved
2	OTCMarketLib.sol: Offers can be partially filled	Critical	Resolved
3	OTCAdminFacet.sol: Modifying the protocol's USDC address can lead to loss of user funds	Major	Resolved
4	OTCMarketFacet.sol: Denial-of-Service might make orders unprocessable	Major	Resolved
5	OTCInitFacet.sol: Discrepancy between OTCInitFacet and OTCAdminFacet admin validation	Minor	Resolved
6	OTCAdminFacet.sol: All admins can potentially get locked out of the contract	Minor	Resolved
7	OTCMarketLib.sol: Function createOTCOffer() passes an incorrect amount unit in InsufficientAmount error	Minor	Resolved
8	OTCMarketLib.sol: User can bypass their allowed supply of vesting tokens	Minor	Resolved
9	OTCMarketFacet.sol: Offers can be processed when contracts are paused	Minor	Acknowledged
10	OTCMarketLib.sol: maxSellableForPosition() function does not account for the 50% limit of vesting positions	Minor	Resolved
11	OTCVestingLib.sol: Function _claim() can	Minor	Resolved

	potentially exceed the block gas limit and revert		
12	OTCVestingLib.sol: Function _splitUpIntervalAtStart() Not checking return values for update in StructuredLinkedList	Minor	Resolved
13	OTCMarketLib.sol: Function createOTCOffer() percentageInBPS can be set 0 during offer creation	Minor	Resolved
14	Mixins.sol: All ACL modifiers produce the same error	Note	Acknowledged
15	OTCAdminFacet.sol: Functions pause() and unpause() do not emit events	Note	Acknowledged
16	OTCAdminFacet.sol: Function addAdmin() does not check if the provided admin already exists	Note	Acknowledged
17	OTCMarketLib.sol: Hardcoded deadline and startTime offer parameters	Note	Acknowledged
18	OTCMarketFacet.sol: Cannot query the percentages for a specific offer	Note	Acknowledged
19	OTCMarketLib.sol: Unconventional createOTCOffer function parameter naming	Note	Acknowledged
20	Unused contract Errors.sol	Note	Acknowledged
21	OTCMarketLib.sol: Inconsistencies in enforcing the deadline	Note	Acknowledged
22	Lack of indexed parameters in events	Note	Acknowledged
23	OTCMarketLib.sol: Supplying and invalid offer ID will revert with an InvalidAmount error	Note	Acknowledged
24	OTCVestingLib.sol: Function _distributeClaim() can potentially leave dust amount in the contract	Note	Acknowledged
25	Miscellaneous	Note	Acknowledged

Critical Issues

1. OTCMarketLib.sol: Arithmetic underflow can lock funds

In the `OTCOffer` struct, the array `filledPerBracket` keeps track of how many treasury, vesting, or staking GFly have been supplied. Without loss of generality, assume many users supply vested GFly exceeding the `maxFillableForBracket`, by a factor of 2. During the execution of the `processOffers()` function, the amount of each proposal is retrieved and to calculate the proportion of each proposal, since the bracket has been overfilled, it is divided by `filledPerBracket` and then multiplied by `maxFillableForBracket`. Since each proposal's sellable amount is less than `filledPerBracket`, the operation will underflow, resulting in 0. Thus, the `claimableUSDC` amount will stay as 0, and sellers will not be able to withdraw their funds.

Recommendation

Multiply sellable by `maxFillableForBracket` first, and then divide by `filledPerBracket`.

Status

Resolved

2. OTCMarketLib.sol: Offers can be partially filled

The `filled` variable of the `Offer` struct keeps track of the total of all the brackets, such that the offer can be processed, otherwise, the funds are returned to the offer creator. However, the filled variable can be overfilled from a single bracket, exceeding the `offer.amountInWei` without all brackets being filled. This could execute partially filled offers, allowing malicious offer creators to claim any available GFly from the OTC contract at `L286` that could exist for the fulfillment of other offers (without the admin having supplied any GFly from the treasury). At the same time,

unsuspected legitimate offer creators could lose part of their `USDC` deposits, for partial fulfillment of their orders.

Recommendation

Modify the logic that adds supplied to `offer.filled` in `supplyToOTCOfferTreasury()`, `supplyToOTCRoundVesting()` and `supplyToOTCRoundStaking()`, such that if a bracket has been overfilled, only the `maxFillablePerBracket` is added to `offer.filled`.

Status

Resolved

Major Issues

3. `OTCAdminFacet.sol`: Modifying the protocol's `USDC` address can lead to loss of user funds

Calling the function `setUSDC()` with a new `USDC` address can lock any existing `USDC` user funds in the contract.

Recommendation

Remove the `setUSDC()` function, as there should be no need to modify this value past contract initialization.

Status

Resolved

4. OTCMarketFacet.sol: Denial-of-Service might make orders unprocessable

---

The function `processOffers()` in `OTCMarketFacet.sol` is called by BattleFly bots to settle over-the-counter offers. To do so, the function iterates over all `offersToProcess` in an unbounded loop. Due to its computational complexity, if there are many offers submitted in the same timeframe, and a significant amount of proposals for each, the function would hit the gas block limit, denying service. A DOS attack can target specific orders, making them unprocessable in the 1-day window available. It is worth noting that since `priceInUSDCPerUnit` is in `USDC` denominations, the cost of creating an offer is very low. Accounting for the current total supply of GFly, the minimum amount that needs to be deposited is around 16000 USDC denominations.

Recommendation

Add a function `processOffer(offerId)` to allow the Battlefly bots to execute each offer separately, if the need arises. Refactor the code to account for the `USDC` decimal points, increasing drastically the amount of deposit for an offer creation and subsequently the cost of attack. Set a reasonable floor for the amount to be supplied in each proposal, i.e., the `toSell` variable in functions `supplyToOTCRoundVesting()` and `supplyToOTCRoundStaking()`.

Status

Resolved

Minor Issues

5. OTCInitFacet.sol: Discrepancy between OTCInitFacet and OTCAdminFacet admin validation

Function `OTCAdminFacet.addAdmin()` validates `admin` against `address(0)`, while `OTCInitFacet.initialize()` fails to provide the same validation.

Recommendation

Validate `admin` in `OTCInitFacet.initialize()`.

Status

Resolved

6. OTCAdminFacet.sol: All admins can potentially get locked out of the contract

The function `removeAdmin()` allows all the contract admins to be removed, thus potentially reaching a state where no admins are left.

Recommendation

Prevent an admin from removing their own account in `removeAdmin()`.

Status

Resolved

7. OTCMarketLib.sol: Function createOTCOffer() passes an incorrect amount unit in InsufficientAmount error

---

The function `createOTCOffer()` passes an amount in the `InsufficientAmount` error that is in different units from the amount the user had passed to the function.

Recommendation

Pass `amountInEth` in `InsufficientAmount` instead of `amountInWei`.

Status

Resolved

8. OTCMarketLib.sol: User can bypass their allowed supply of vesting tokens

---

The function `supplyToOTCRoundVesting()` allows users to supply an offer from a vesting position. There is a limit of half of the tokens to be vested in the next 12 months, however the limit can be by bypassed by calling the function multiple times since `maxSellable` is calculated by `(totalClaimable - ms.reservedForProposalsPerBracket[1][vestingId]) / 2`. Since `totalClaimable` is constant and `ms.reservedForProposalsPerBracket[1][vestingId]` increase by `maxSellable` a user can supply half of it's remaining to-be-vested tokens each time, until they virtual have a single token denomination.

Recommendation

Revert if `ms.pendingProposalsPerBracket[1][vestingId] == true`.

Status

Resolved

9. OTCMarketFacet.sol: Offers can be processed when contracts are paused

---

The functions `processOffer()` and `processOffers()` can be executed by the Battlefly bots even when the contracts are paused.

Recommendation

Restrict the processing of offers when the contracts are paused by adding the `whenNotPaused` modifier to the function.

Note

The same issue exists for `setPercentagesForOffer()` and `setOfferPercentages()` in `OTCMarketFacet.sol` and all `onlyAdmin` functions in `OTCAdminFacet.sol`.

Status

Acknowledged. Team's response: "*Not necessary to fix as this function can only be called by the BattleFly bots, owned by the core team*".

10. OTCMarketLib.sol: maxSellableForPosition() function does not account for the 50% limit of vesting positions

---

The function `maxSellableForPosition()` is used to query the amount a user can supply to OTC offers from staking or vesting positions. However, for vesting positions, the limit of 50% for the next 12 months is not applied and the whole claimable amount for the next 12 months is returned, which is misleading.

Recommendation

Consider returning 50% of the amount calculated if `positionType == 1`.

Status

Resolved

11. OTCVestingLib.sol: Function _claim() can potentially exceed the block gas limit and revert

---

Function `_claim()`'s while loop can run indefinitely until it exceeds the block gas limit if block.timestamp is identical to interval.start. The while loop only stops when next is 0 but if block.timestamp is equal to interval.start, if-else ladder condition will be false and it will run infinitely.

Recommendation

Change the last condition of the if-else ladder from if (block.timestamp > interval.start) to if (block.timestamp >= interval.start).

Status

Resolved

12. OTCVestingLib.sol: Function _splitUpIntervalAtStart() Not checking return values for update in StructuredLinkedList

---

Function `_splitUpIntervalAtStart()`, an update is made in StructuredLinkedList distributionIntervalIds, but there is no check on whether the update is successful. For instance, there is no check for the return value of insertAfter() and it is ignored.

Recommendation

Revert the call if an update is not successful instead of failing silently.

Note

Similar issues exist in functions _splitUpIntervalAtEnd() and _splitUpIntervalAtStartAndEnd().

Status

Resolved

13. OTCMarketLib.sol: Function createOTCOffer() percentageInBPS can be set 0 during offer creation

---

Function `createOTCOffer()` can be called before the admin has set percentageInBPS, in which case percentageInBPS for the offer created will be set to 0.

Recommendation

Before the creation of an offer, implement a check to validate if percentageInBPS is non-zero.

Status

Resolved

Informational Notes

14. Mixins.sol: All ACL modifiers produce the same error

All ACL modifiers in the `WithACLModifiers()` revert with the exact same error, which can make it challenging for users to determine why their transaction has been reverted.

Recommendation
Consider either creating a separate error for each different type of ACL restriction, or adding a parameter to `AccessDenied()` that specifies the restriction type.

Status
Acknowledged

15. OTCAdminFacet.sol: Functions pause() and unpause() do not emit events

Recommendation
Consider having the aforementioned functions emit the appropriate events so that protocol participants can more conveniently detect when the contracts have been paused/unpaused.

Note
The same issue exists in functions: `setGFly()`, `setVestedGFly()`, `setUSDC()` and `setGFlyStaking()`.

Status
Acknowledged

16. OTCAdminFacet.sol: Function addAdmin() does not check if the provided admin already exists

Failing to check that the provided `admin` already exists can result in an `AdminAdded` event being emitted when no actual `admin` has been added.

Recommendation

Consider reverting if `OTCAdminStorage.layout().admins[admin] == true`.

Note

Similar issues exist in functions: `removeAdmin()`, `removeVestingManager()`, `addVestingManager()`, `removeBattleflyBot()`, `addBattleflyBot()`, `addPauseGuardian()`, and `removePauseGuardian()`.

Status

Acknowledged

17. OTCMarketLib.sol: Hardcoded deadline and startTime offer parameters

The function `createOTCOffer()` creates an offer that has a hardcoded deadline of `block.timestamp + 7 days`, and a `startTime` of `block.timestamp + 8 days`. Enforcing parameters can potentially make the contract inflexible and unable to adjust to future needs.

Recommendation

Consider implementing a setter function for those parameters, that would give the admin some flexibility, in case it is required in the future.

Status

Acknowledged

18. OTCMarketFacet.sol: Cannot query the percentages for a specific offer

---

The function `getOfferPercentages()` returns the default offer percentages stored in `OTCMarketStorage`, but the admin can change the percentages for a specific offer. Those cannot be queried and returning the default offer percentages might be misleading for users interested in the percentages of the specific offer.

Recommendation

Consider implementing a `view` function `getOfferPercentages(offerId)`.

Status

Acknowledged

19. OTCMarketLib.sol: Unconventional createOTCOffer function parameter naming

---

Function parameter `amountInEth` name can mislead users.

Recommendation

Consider renaming the function parameter to `amount` and document that this amount should not account for decimal points.

Status

Acknowledged

20. Unused contract Errors.sol

The contract `Errors.sol`, defining errors, is currently unused.

Recommendation

Consider removing the contract if it is not planned to be used in the future.

Status

Acknowledged

21. OTCMarketLib.sol: Inconsistencies in enforcing the deadline

---

Users should be able to supplyToOTCRoundVesting() unless the deadline of the offer has passed. However, if the block.timestamp is equal to the deadline, the proposal is rejected.

Recommendation

Consider rejecting the proposal only if block.timestamp > offer.deadline.

Note

The same exists in supplyToOTCRoundStaking().

Status

Acknowledged

22. Lack of indexed parameters in events

Most of the contracts' event parameters are not indexed.

Recommendation

Consider indexing event parameters to help make searching and filtering for specific events more convenient.

Status

Acknowledged

23. OTCMarketLib.sol: Supplying and invalid offer ID will revert with an InvalidAmount error

---

Calling the function supplyToOTCRoundVesting() with an invalid offerId, will not immediately fail, but will revert with an InvalidAmount error in OTCMarketLib.sol:L149.

Recommendation

Consider reverting with an appropriate error message if the offer cannot be found.

Note

The same issue exists in supplyToOTCRoundStaking(), and in setPercentagesForOffer(), where an invalid offerId will cause a revert with DeadlineOfferReached().

Status

Acknowledged

24. OTCVestingLib.sol: Function _distributeClaim() can potentially leave dust amount in the contract

---

Function `_distributeClaim()` distributes GFly to all interval owners according to their ownership percentage. Due to rounding errors, a small amount can potentially be left unclaimable.

Recommendation

Consider assigning the remaining dust to the last owner.

Status

Acknowledged

25. Miscellaneous

The following are some recommendations to improve the code quality and readability:

- Unnecessary imported library `EnumerableSet` in `facets/OTCMarketFacet.sol`: `L7`.
- Unnecessary imported libraries `DiamondOwnable`, `StructuredLinkedList`, `EnumerableSet` in `facets/OTCVestingFacet.sol`.

Status

Acknowledged

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of BattleFly or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH