SOLIDIFIED

Audit Report for BattleFly - July 12, 2022

Summary

Audit Report prepared by Solidified covering the BattleFly 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 July 12, 2022, and the results are presented here.

Audited Files

The source code has been supplied in a compressed ZIP format with the following SHA256 hash: `1dbcebbc032e8064ce14ac08f4b79c74938383d0a7a14c22205ba8cfbba72904`

Update: The BattleFly team provided an updated ZIP file with issue fixes on July 15, 2022, with the following SHA256 hash:
`00e6aaf465a7dac2b9776e0c6ccd5cd2aa27772335f2eb0c03bc78e1828e0597`

File list:

```
├── BattleflyAtlasStakerV02.sol
├── BattleflyFlywheelVaultV02.sol
├── interfaces
│   ├── IAtlasMine.sol
│   ├── IBattlefly.sol
│   ├── IBattleflyAtlasStakerV02.sol
│   ├── IBattleflyFlywheelVaultV02.sol
│   ├── IMasterOfCoin.sol
│   └── IVault.sol
└── libraries
    └── BattleflyAtlasStakerUtils.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	High	-
Level of Documentation	Medium	-
Test Coverage	High	-

SOLIDIFIED

Audit Report for BattleFly - July 12, 2022

Issues Found

Solidified found that the BattleFly contracts contain no critical issues, 2 major issues, 4 minor issues, and 6 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	BattleflyAtlasStakerV02.sol: Vault users can potentially lose funds after admin calls removeVault()	Major	Resolved
2	BattleflyAtlasStakerUtils.sol: Incorrect amount of weeks used as a period of six months	Major	Resolved
3	BattleflyAtlasStakerV02.sol: Vault users can permanently lose access to their funds if BattleflyAtlasStakerV02 is paused indefinitely	Minor	Resolved
4	BattleflyAtlasStakerV02.sol: Inefficient usage of the retention period when requesting withdrawal	Minor	Resolved
5	BattleflyAtlasStakerV02.sol: Daily executeAll() cronjob puts lots of trust in Battlefly	Minor	Resolved
6	BattleflyFlywheelVaultV02.sol: Function withdraw() can save on gas by declaring _depositIds as calldata	Note	Resolved
7	BattleflyFlywheelVaultV02.sol: Parameter _magic is redundant in function initialize()	Note	Resolved
8	BattleflyAtlasStakerV02.sol: The owner can set arbitrary high fees when adding a new vault	Note	Acknowledged
9	Use of floating pragma	Note	Acknowledged
10	Misc Notes	Note	Resolved
11	Misc Gas Optimizations	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

1. BattleflyAtlasStakerV02.sol: Vault users can potentially lose funds after admin calls removeVault()

Functions `withdraw()`, `requestWithdrawal()` and `claim()` all require that the calling vault must be whitelisted. In case the contract owner delists a vault that had made deposits, all vault users will have their funds locked in the contract with no way to withdraw them.

Recommendation

Remove the `onlyWhitelistedVaults` modifier from functions `withdraw()`, `requestWithdrawal()` and `claim()`, thus allowing delisted vaults to still withdraw their deposited funds.

Status

Resolved

2. BattleflyAtlasStakerUtils.sol: Incorrect amount of weeks used as a period of six months

The function `getLockPeriod` returns the lock period in days depending on the given parameter `_lock`. However, for a `_lock` value of `IAtlasMine.Lock.sixMonths`, `23 weeks` does not add up to the anticipated six months lock period. Considering a month has roughly `4.34524` weeks, six months equals roughly `26 weeks`.

A locking period shorter than the anticipated 6 months will cause issues when a user tries to withdraw, even though funds are still staked in Atlas staker and not ready for withdrawal.

Recommendation

Use `26 weeks` or, consider using `days` instead of weeks for more accuracy.

Status

Resolved

Minor Issues

3. BattleflyAtlasStakerV02.sol: Vault users can permanently lose access to their funds if BattleflyAtlasStakerV02 is paused indefinitely

---

There is always a risk that `BattleflyAtlasStakerV02` gets paused indefinitely, in case, for instance, the owner lost their private keys after pausing the contract. In such a case, users will permanently not be able to withdraw their funds.

Recommendation

Either eliminate the pausing functionality or set a maximum amount of time that the contract could be paused for.

Status

Resolved

4. BattleflyAtlasStakerV02.sol: Inefficient usage of the retention period when requesting withdrawal

---

Deposited funds are restaked in blocks of 14 days after unlocking. To withdraw deposits, a user has to request withdrawals a certain amount of time in advance. After a retention period of 14 days, funds can then be withdrawn. However, requesting withdrawals prior to the locking period end does not require a retention period. Additionally, the current implementation naively sets the `retentionUnlock` to `currentEpoch + 14` days which is longer than needed:

Given a deposit with `unlockAt` set to epoch `28`. A withdrawal request can be made as early as in epoch `14`. The user requests the withdrawal in epoch `20`, which sets retentionUnlock to epoch `34`. Fast forward to epoch `28`, funds are automatically withdrawn from Atlas Mine and immediately restaked due to the retentionUnlock set to epoch `34`, which extends the earliest possible withdrawal to epoch `42` instead of an earlier possible withdrawal in epoch `28`.

Recommendation

Consider omitting the retention period for withdrawal requests before the unlocking period ends. For withdrawal requests after the unlocking period ended, use the next possible unlocking date (based on the 14 days restaking period).

Status

Resolved

5. BattleflyAtlasStakerV02.sol: Daily executeAll() cronjob puts lots of trust in BattleFly

---

The `executeAll()` function is intended to be called once a day by the Battlefly `BATTLEFLY_BOT`. The current implementation has no protection against running this function multiple times a day and puts full trust into the `BATTLEFLY_BOT` to run as expected once a day. The `executeAll()` function is integral to the protocol as it handles reward harvests, withdrawals and deposits as well as starting a new epoch (epoch = 1 day).

However, if executed multiple times a day, epochs are tracked incorrectly and will cause timing issues with the `Atlas Mine` contract. Additionally, users have to trust a central authority to call this function. If, for any reason, this function is not called daily, rewards can be forfeit and withdrawals are not possible.

Recommendation

Consider implementing a check to prevent running the `executeAll()` function multiple times a day. Further, consider making the function publicly callable by everyone.

Status

Resolved

Informational Notes

6. BattleflyFlywheelVaultV02.sol: Function withdraw() can save on gas by declaring _depositIds as calldata

---

Declaring the parameter `_depositIds` as `calldata` instead of `memory` can potentially save on gas, since the array values would be directly read from `calldata` instead of being copied to `memory` first.

Recommendation

Declare the `_depositIds` parameter as `calldata` instead of `memory`.

Note

The same issue exists in functions: `requestWithdrawal()`, `whitelistUsers()` and `removeUsers()`.

Status

Resolved

7. BattleflyFlywheelVaultV02.sol: Parameter _magic is redundant in function initialize()

The `_magic` parameter is redundant since it can be retrieved from `_atlasStaker`.

Recommendation

Remove the redundant parameter to prevent any discrepancies between the `BattleflyFlywheelVaultV02.MAGIC` and `BattleflyAtlasStakerV02.MAGIC` addresses.

Status
Resolved

8. BattleflyAtlasStakerV02.sol: The owner can set arbitrary high fees when adding a new vault

---

While adding a new vault, the owner can specify a fee rate `fee`. On line 42, the comment implies a max fee of 10% (*"Max fee the owner can ever take - 10%"*). However, there is no upper fee boundary implemented. Thus, the owner can set any arbitrary high fee for a vault.

Recommendation
Consider adding a reasonable upper bound for `fee` in `addVault`.

Status
Acknowledged. Team's response: "*We removed the comment in code. Fees can be set potentially higher in the future*".

9. Use of floating pragma

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

https://swcregistry.io/docs/SWC-103

Recommendation

Lock the pragma version in all contracts and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

Status

Acknowledged. Team's response: "*We like to keep the flexibility of new features of solidity if necessary + we have extensive test coverage*".

10. Misc Notes

---

1. Events not indexed. Findings:

- `BattleflyFlywheelVaultV02.sol#L279` - `event NewUserStake(uint256 depositId, uint256 amount, uint256 unlockAt, address owner, IAtlasMine.Lock lock);`. Resolved.
- `BattleflyFlywheelVaultV02.sol#L280` - `event UpdateUserStake(uint256 depositId, uint256 amount, uint256 unlockAt, address owner, IAtlasMine.Lock lock);`. Resolved.
- `BattleflyFlywheelVaultV02.sol#L285` - `event AddedUser(address vault);`. Resolved.
- `BattleflyFlywheelVaultV02.sol#L286` - `event RemovedUser(address vault);`. Resolved.

2. Code not used. Findings:

- `IBattlefly.sol` is imported in `BattleFlywheelVaultV02` contract but it is not used. Resolved.

11. Misc Gas Optimizations

---

1. `BattleflyFlywheelVaultV02.sol`: Unused `PausableUpgradeable` contract. Consider removing it. Resolved.
2. `BattleflyAtlasStakerV02._executeDepositAll`: Reset `unstakedAmount[LOCKS[i]]` to 0 only if amount has been `> 0`. Resolved.
3. `BattleflyAtlasStakerV02._deposit`: Redundant assignment to variable `vaultStake.amount`. Consider removing the assignment on line 626. Resolved.

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