SOLIDIFIED

Audit Report for Battlefly Deposit Locker - January 11, 2023

Summary

Audit Report prepared by Solidified covering the Battlefly Deposit Locker 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 January 11, 2023, and the results are presented here.

Audited Files

The source code has been supplied in a zip file.

Zip file sha256 hash:
`c14c67f1fba42c9c6154192a84eab8c2cc21131b36c91f42436987f31594c810`

Update: Fixes were received on Thursday January 19, 2023.
Updated Zip file sha256 hash:
`bff1f7e50821f3d3fb3776fa39778e45b9b09c10f8a774b46c02e8ed92230fd4`

File list:

```
./deposit-locker
├── acl
│   ├── ACLAdminFacet.sol
│   ├── ACLStorage.sol
│   ├── DiamondOwnable.sol
│   └── IACLAdmin.sol
├── deposit
│   ├── DepositLockerAdminFacet.sol
│   ├── DepositLockerLib.sol
│   ├── DepositLockerStorage.sol
│   ├── DepositLockerUsersFacet.sol
│   ├── IDepositLockerAdmin.sol
│   └── IDepositLockerUsers.sol
├── diamond
│   ├── IDiamondCut.sol
│   └── LibDiamond.sol
├── init
│   └── InitFacet.sol
```

```
├── migration
│   ├── DepositMigrationFacet.sol
│   ├── DepositMigrationStorage.sol
│   ├── IDepositMigration.sol
│   └── VaultAdapter.sol
└── utils
    └── Mixins.sol
```

Intended Behavior

Battlefly Deposit Locker is a system for general management of user deposits for the Battlefly game.

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	Low	-
Code readability and clarity	High	-
Level of Documentation	Medium	-
Test Coverage	High	-

SOLIDIFIED

Audit Report for Battlefly Deposit Locker - January 11, 2023

Issues Found

Solidified found that the Battlefly Deposit Locker contracts contain 0 critical issues, 1 major issue, 3 minor issues, 8 informational notes and 2 warnings.

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	DepositLockerAdminFacet.sol: Calling function setMagic() with a new magic_ value can lead to loss of user funds	Major	Acknowledged
2	DepositLockerUsersFacet.sol: Users can permanently lose access to their funds if DepositLockerUsersFacet is paused indefinitely	Minor	Acknowledged
3	DepositLockerAdminFacet.sol: There is no way to remove or deactivate a LockOption	Minor	Resolved
4	LockInDays can be set arbitrarily high	Minor	Resolved
5	ACLAdminFacet.sol: Functions pause() and unpause() do not emit events	Note	Resolved
6	ACLAdminFacet.sol: Function addAdmin() does not check if the provided admin already exists	Note	Resolved
7	DiamondOwnable.sol: Function transferOwnership() Single step ownership transfer can be dangerous	Note	Acknowledged
8	DepositLockerUsersFacet.sol Function withdraw() does not give a meaningful error when amountToWithdraw exceeds liquidBalance	Note	Acknowledged
9	Lack of indexed parameters in events	Note	Resolved
10	LibDiamond.sol is outdated	Note	Acknowledged

11	Gas Optimization	Note	Acknowledged
12	Miscellaneous	Note	Resolved
13	Fund managers have unrestricted access to user funds	Warning	Acknowledged
14	DepositLockerLib.sol: Function updateWithdrawableAmount() can potentially exceed the block gas limit	Warning	Acknowledged

Critical Issues

No critical issues have been found.

Major Issues

1. DepositLockerAdminFacet.sol: Calling function setMagic() with a new magic_ value can lead to loss of user funds

Calling the function `setMagic()` with a new `magic_` address after users had already deposited funds to the old address will result in users not being able to withdraw their deposited funds.

Recommendation

The value of `DepositLockerStorage.layout().magic` should be immutable if users have any existing `magic` deposits.

Status

Acknowledged. Team's response: "*We opted to keep setMagic in as token migrations are a thing and it can only be done by an admin, which will be the Battlefly DAO Multisig (3/7 signatures required)*".

Minor Issues

2. DepositLockerUsersFacet.sol: Users can permanently lose access to their funds if DepositLockerUsersFacet is paused indefinitely

There is always a risk that `DepositLockerUsersFacet` gets paused indefinitely, for instance, if 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.

Note

The same issue exists for `DepositLockerAdminFacet`, where fund managers are not able to return user funds if `DepositLockerAdminFacet` is paused indefinitely.

Status

Acknowledged. Team's response: "*We consider this a very low risk as new pausing guardians can always be added by the Multisig admin. We have 7 signers on that multisig with 3 signatures required. Therefore we kept it as is*".

3. DepositLockerAdminFacet.sol: There is no way to remove or deactivate a LockOption

---

The contract does not provide the admin with any way to remove an added `LockOption` or set its `active` variable to `false`.

Recommendation

Implement `removeLockOption()` and/or `deactivateLockOption()` function(s).

Status

Resolved

4. LockInDays can be set arbitrarily high

A `LockOption` that is erroneously set up, having an extremely high value would cause `requestWithdrawal` to always revert during safe-casting to uint32, upon calculation of the `unlockDay`. This will make it impossible for users with deposits on that `LockOption` to `requestWithdrawal`.

Recommendation

Consider enforcing reasonable limits when setting up a `LockOption` with `addLockOption`.

Status

Resolved

Informational Notes

5. ACLAdminFacet.sol: Functions pause() and unpause() do not emit events

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

Status
Resolved

6. ACLAdminFacet.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 `ACLStorage.layout().admins[admin] == true`.

Note
Similar issues exist in the following functions: `removeAdmin()`, `addFundManager()`, `removeFundManager()`, `addPauseGuardian()`, and `removePauseGuardian()`.

Status
Resolved

7. DiamondOwnable.sol: Function transferOwnership() Single step ownership transfer can be dangerous

---

Single-step ownership transfer can potentially lose contract ownership if the wrong address is provided.

Recommendation

Consider using the two-step ownership transfer pattern. If ownership renouncement is a requirement, consider providing a separate function for that.

Status

Acknowledged. Team's response: "*We have 7 people to confirm the ownership transfer as it's a multisig*".

8. DepositLockerUsersFacet.sol: Function withdraw() does not give a meaningful error when amountToWithdraw exceeds liquidBalance

---

While withdraw() will indeed revert if amountToWithdraw exceeds liquidBalance, the user will only get a generic error that does not provide meaningful information.

Recommendation

Consider providing a more contextually appropriate error message when the aforementioned case is true.

Status

Acknowledged. Team's response: "*We cover this on our frontend. The contracts can be used without, but end users will normally always work with the frontend*".

9. 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

Resolved

10. `LibDiamond.sol` is outdated

The implementation of `LibDiamond.sol` is more than one and a half years old, with several minor improvements being introduced by the author that improve the overall code quality and introduce gas optimisations.

Recommendation

Consider using the latest version of `LibDiamond.sol` by Nick Mudge: https://github.com/mudgen/diamond-3-hardhat/blob/main/contracts/libraries/LibDiamond.sol

Status

Acknowledged. Team's response: "*We work with the Hardhat plugin which uses this Diamond version. Will always look for the latest working version compatible with Hardhat (while using Hardhat)*".

11. Gas Optimization

The EVM operates on 256-bit storage slots, thus smaller lengths need to be padded and unpadded costing more gas. It is more efficient to pack variables in structures in 256-bits and rearrange variables in such a way that the number of storage slots are minimized.

Recommendation

Consider rearranging the variables in structs `UserStakeV1`, `UserStakeV2`, `UserDepositState` in `VaultAdapter.sol`, such that:

- `UserStakeV1`: `lock` and `owner` are packed together.
- `UserStakeV2`: `lockAt`, `owner`, and `lock` are packed together.
- `UserDepositState`: `owner` and `withdrawlRequested` are packed together.

Status

Acknowledged. Team's response: "*We consider this minor as we are working on Arbitrum where gas costs are close to negligible*".

12. Miscellaneous

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

- Unnecessary usage of library in `DepositLockerAdminFacet.sol`: `L21` and `DepositLockerUsersFacet.sol`: `L17`
- Function parameter `magic` in `IDepositLockerAdmin.sol`: `setMagic` shadows `IDepositLocker`: `magic`. **Status**: Resolved.

Warnings

13. Fund managers have unrestricted access to user funds

Fund managers are able to withdraw all user funds without any guarantees that these funds will be returned back to the protocol.

Status
Acknowledged. Team's response: "*This is per design and will be managed with a multisig. We need this as the funds are used in the Harvester war contracts which are difficult to manage by contracts only*".

14. `DepositLockerLib.sol`: Function `updateWithdrawableAmount()` can potentially exceed the block gas limit

The function `updateWithdrawableAmount()` is used to delete withdrawals that have reached maturity date from `withdrawalsByDay` and add its amount to `withdrawableDeposits`. The updating of storage in a loop is costly, and whilst this loop is bound by the days the amount has been stale, if the function is not called for a prolonged period of time, the gas needed performing the computation for several hundreds of days will hit the block gas limit. This will lock deposits in the contract, as any attempt to `requestWithdrawal()`, `withdraw()` or `adminWithdraw()` will be denied. It is also worth noting that since this calculation is performed on user's `requestWithdrawal` or `withdraw` functions, this can cause an unfairly high amount of gas for users that attempt to withdraw, after long periods of withdrawable amounts being stale.

Status

Acknowledged. Team's response: "*We consider this minor as we are working on Arbitrum where gas costs are close to negligible + the gas limits are lots higher than on mainnet. If ever needed we can always upgrade the diamond for it*".

Audit Report for Battlefly Deposit Locker - January 11, 2023

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