SOLIDIFIED

Audit Report for Pandora - May 19, 2022

Summary

Audit Report prepared by Solidified covering the Pandora 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 May 9, 2022, and the results are presented here.

Audited Files

The source code has been supplied in a private source code repository:

https://github.com/advancedblockchain/pandora-protocol

Commit number: `3ed61810ccd86b37ac6967b03a775fd35c904d5f`

Update: Fixes where provided on May 12, 2022
Updated commit number: `0bcbd3159d36b8de28c04d546489246c371bd4c7`

Intended Behavior

Pandora is a lending protocol where the collateral can be freely swapped and invested into different whitelisted protocols and tokens.

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	High	-
Test Coverage	High	-

SOLIDIFIED

Audit Report for Pandora - May 19, 2022

Issues Found

Solidified found that the Pandora contracts contain no critical issues, 8 major issues, 9 minor issues, and 11 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	StablesManager.sol: Replacing the current pandoraUSD token can lead to loss of funds for all existing borrowers	Major	Resolved
2	SharesRegistry.sol: Contract owner can liquidate all Pandora borrowers by assigning a malicious oracle	Major	Resolved
3	AaveStablecoin.sol: Contract owner can potentially drain deposited funds	Major	Resolved
4	StrategyManager.sol: Pandora users could potentially lose funds when a strategy is removed	Major	Resolved
5	SharesRegistry.sol: Anyone can call accrue() and manipulate earned fees by protocol	Major	Resolved
6	SharesRegistry.sol: Unable to accrue interest as INTEREST_PER_SECOND is not set and can not be updated	Major	Resolved
7	YearnStablecoin.sol: Fees for accumulated rewards will always be 0	Major	Resolved
8	OperationsLib.sol: safeApprove() is missing approve(0)	Major	Resolved
9	HoldingManager.sol: Function createHoldingForMyself() does not enforce the firstDepositAmount requirement	Minor	Acknowledged

10	HoldingManager.sol: The pause functionality is ineffective	Minor	Resolved
11	SharesRegistry.sol: Function acceptOwnership() does not reset the value of temporaryOwner	Minor	Resolved
12	Holding.sol: Unchecked low-level call() result	Minor	Resolved
13	ChainlinkOracle.sol: Chainlink's latestRoundData might return stale or incorrect results	Minor	Resolved
14	ModChainlinkOracle.sol: Use of deprecated Chainlink oracle aggregator function latestAnswer in _get()	Minor	Resolved
15	PandoraUSD.sol: Owner can mint and burn unlimited tokens	Minor	Resolved
16	Staker.sol: _performanceFee is not validated in the constructor	Minor	Resolved
17	SimpleUniswapV2Oracle.sol: Exchange rate is vulnerable to flash loan attacks	Minor	Resolved
18	HoldingManager.sol: The _isContract() check in function _assignHolding() could be bypassed	Note	Resolved
19	TwapUniswapV2Oracle.sol: Function update() can save on gas if it only fetches the current prices after checking the period	Note	Resolved
20	AaveStablecoin.sol: Function deposit() is missing an AAVE referral code	Note	Resolved
21	lpVault is redundant across all Curve contracts	Note	Resolved
22	SigningManager.sol: Function getDomainSeparator() unnecessarily uses assembly to fetch the current chain id	Note	Resolved
23	SharesRegistry.sol: Oracle might return 0	Note	Resolved
24	PandoraUSD.sol: Use ERC20.totalSupply instead of totalMinted	Note	Resolved

25	Contracts inheriting Ownable have renounceOwnership() functionality	Note	Resolved
26	OperationsLib.sol: Use .selector instead of a hexadecimal number	Note	Resolved
27	Gas Optimizations	Note	-
28	Miscellaneous Notes	Note	-

Critical Issues

No critical issues have been found.

Major Issues

1. StablesManager.sol: Replacing the current pandoraUSD token can lead to loss of funds for all existing borrowers

Function `setPandoraUSD()` allows the contract owner to arbitrarily replace the current `pandoraUSD` token at any point in time. This action will prevent all existing borrowers from settling their debts, since all of them will be still holding the old `pandoraUSD` token, causing the `repay()` function to always revert.

Recommendation

Only allow setting `pandoraUSD` in the contract's constructor.

Status

Resolved

2. SharesRegistry.sol: Contract owner can liquidate all Pandora borrowers by assigning a malicious oracle

Function `setOracleData()` allows the contract owner to potentially assign a malicious (or a buggy) oracle that would allow them (or an attacker) to liquidate all the current protocol borrowers.

Recommendation

`setOracleData()` should not be able to immediately reassign the oracle data, but rather give market participants adequate time to close their positions (in case they wish to) before a new oracle is assigned.

Status

Resolved

3. AaveStablecoin.sol: Contract owner can potentially drain deposited funds

---

Function `setLendingPool()` allows the contract owner to potentially assign a malicious `lendingPool` contract at any arbitrary point in time, thus potentially allowing them to drain all newly deposited funds.

Recommendation

Only allow setting `lendingPool` in the contract's constructor.

Note

The same vulnerability exists in the following contracts: `CurveBase`, `Curve3Pool`, `CurveDaiUsdcUsdtPool`, and `CurveTricrypto`.

Status

Resolved

4. StrategyManager.sol: Pandora users could potentially lose funds when a strategy is removed

Function `removeStrategy()` allows the contract owner to remove any strategy regardless of its current usage state. Removing a strategy that's in use will prevent its respective users from calling function `claimInvestment()`, and thus lead to potential loss of funds for the protocol users.

Recommendation

Keep track of what strategies are in use and only allow removal of unused strategies.

Status

Resolved

5. SharesRegistry.sol: Anyone can call accrue() and manipulate earned fees by protocol

The `accrue()` function handles the accumulation of fees. The function lacks access control and can be called by anyone. A malicious caller can call the function and manipulate the amount of earned fees by the protocol.

Recommendation

Use the modifier `onlyStableManager` to only allow `StableManager` to call `accrue()`.

Status

Resolved

6. SharesRegistry.sol: Unable to accrue interest as INTEREST_PER_SECOND is not set and can not be updated

The `accrue()` function in `SharesRegistry.sol` calculates the accrued interest on the borrowed tokens based on `accrueInfo.INTEREST_PER_SECOND`. But `accrueInfo.INTEREST_PER_SECOND` is never initialized (default set to `0`) nor is there a function to update the value. Hence, borrowing is interest free.

Recommendation

Initialize `accrueInfo.INTEREST_PER_SECOND` to a value `> 0` or consider adding a setter function to allow the owner to update the value.

Status

Resolved

7. YearnStablecoin.sol: Fees for accumulated rewards will always be 0

In the `YearnStablecoin` strategy contract, whenever deposited funds are withdrawn, the protocol intends to take a fee of the accumulated rewards. But due to calling `withdraw()` before calculating the rewards in `getRewards()`, the rewards will always be `0`.

Recommendation

Calculate `rewardPortion` before withdrawing funds.

Status

Resolved

8. OperationsLib.sol: safeApprove() is missing approve(0)

Some tokens, like `USDT` (see requirement line 199, https://etherscan.io/address/0xdac17f958d2ee523a2206206994597c13d831ec7#code), require first reducing the address allowance to `0` by calling `approve(spender, 0)` and then approve the actual allowance.

When using one of these unsupported tokens, all transactions will revert and the protocol cannot be used.

Recommendation

Approve with a zero amount first before setting the actual amount.

Status

Resolved

Minor Issues

9. HoldingManager.sol: Function createHoldingForMyself() does not enforce the firstDepositAmount requirement

---

Function `createHoldingForMyself()` allows users to create and assign a holding for themselves without depositing the `firstDepositAmount`.

Recommendation

Transfer `firstDepositAmount` to the newly assigned holding in order to satisfy the protocol first deposit requirement.

Status

Acknowledged. Team's response: "*That's the intended behavior. A deposit is only required when assigning a holding*".

10. HoldingManager.sol: The pause functionality is ineffective

---

In case of an emergency, the contract's owner can pause the contract, but cannot save users' funds due to the lack of any fund recovery mechanism in the contract.

Recommendation

Either provide a way for the owner to save the funds, or remove the pause functionality.

Status

Resolved

11. SharesRegistry.sol: Function acceptOwnership() does not reset the value of temporaryOwner

After the new owner accepts ownership, function `acceptOwnership()` does not reset the value of `temporaryOwner`.

Recommendation

Set the value of `temporaryOwner` to `address(0)` in order to reflect the correct state of the contract.

Status

Resolved

12. Holding.sol: Unchecked low-level call() result

If the return value of a low-level `call` is not checked, the execution may resume even if the function call throws an error. This can lead to unexpected behavior and inconsistent states.

The following contracts call `genericCall()` but do not check for the `success` return value: `CompoundStablecoin`, `ConvexBase`, `Curve3PoolBase`, `CurveDaiUsdcUsdtPoolBase`, and `CurveTricryptoBase`.

Recommendation

Check the `success` return value of `genericCall()` in the specified contracts and revert if `false`.

Status

Resolved

13. ChainlinkOracle.sol: Chainlink's latestRoundData might return stale or incorrect results

In `ChainlinkOracle.sol`, `latestRoundData()` is used but there is no check if the return value indicates stale data. This could lead to stale prices according to the Chainlink documentation:

https://docs.chain.link/docs/historical-price-data/#historical-rounds
https://docs.chain.link/docs/faq/#how-can-i-check-if-the-answer-to-a-round-is-being-carried-over-from-a-previous-round

Recommendation

Consider adding checks for stale data. For instance:

```
(uint80 roundId, int256 priceC, , uint256 timestamp, uint80 answeredInRound)
= IAggregator(multiply).latestRoundData();

require(priceC > 0, "PRICE: NEGATIVE");
require(answeredInRound >= roundId, "PRICE: STALE PRICE");
require(timestamp != 0, "PRICE: ROUND INCOMPLETE");
```

Status

Resolved

14. ModChainlinkOracle.sol: Use of deprecated Chainlink oracle aggregator function latestAnswer() in _get()

According to Chainlink's documentation, the `latestAnswer()` function is deprecated. This function does not error if no answer has been reached but returns 0.

The function is not present in the latest API reference:

https://docs.chain.link/docs/price-feeds-api-reference/

Recommendation

Use the `latestRoundData()` function to get the price instead. Add checks on the return data with proper revert messages if the price is stale or the round is incomplete.

Status

Resolved

15. PandoraUSD.sol: Owner can mint and burn unlimited tokens

The minting and burning of `PandoraUSD` should only be done by the stables manager.

Recommendation

Change the mint and burn modifier from `onlyOwnerOrStablesManager` to `onlyStablesManager`.

Status

Resolved

16. Staker.sol: _performanceFee is not validated in the constructor

The parameter `_performanceFee` in the constructor is not validated to make sure it is less than `FEE_FACTOR`.

Recommendation

Use `require(_fee < OperationsLib.FEE_FACTOR, "3018");` (from `setPerformanceFee()`) to validate `_performanceFee` in the constructor.

Status

Resolved

17. SimpleUniswapV2Oracle.sol: Exchange rate is vulnerable to flash loan attacks

---

The latest exchange rate is calculated using the current Uniswap token pair reserves. These reserves can be easily manipulated via flash loan attacks to increase the current token balance for the length of the current transaction.

Recommendation

We recommend not using the oracle contract `SimpleUniswapV2Oracle`, or to at least ensure no actor can change the Uniswap pool contents earlier in the same block:

```
(uint256 r0, uint256 r1, uint32 blockTimestampLast) =
IUniswapV2Pair(pair).getReserves();
require(blockTimestampLast < block.number, "Pool changed in this block");
```

Status

Resolved

Informational Notes

18. HoldingManager.sol: The _isContract() check in function _assignHolding() could be bypassed

Any contract address can be precomputed before the contract is actually deployed to it. If a precomputed contract address is passed to function `_assignHolding()`, it will be indistinguishable from an EOA address, and the check will be bypassed.

Recommendation
Remove the redundant check.

Status
Resolved

19. TwapUniswapV2Oracle.sol: Function update() can save on gas if it only fetches the current prices after checking the period

Function `update()` only requires the values fetched from `UniswapV2OracleLibrary.currentCumulativePrices()` if the elapsed time has exceeded `period`, but is currently fetching prices regardless of that condition.

Recommendation
Only call `UniswapV2OracleLibrary.currentCumulativePrices()` if `(timeElapsed >= period)`.

Status
Resolved

20. AaveStablecoin.sol: Function deposit() is missing an AAVE referral code

Function `deposit()` does not provide an AAVE referral code when calling `lendingPool.deposit()`.

Recommendation

Implement a referral code setter function in the contract, so that a referral code is active as soon as AAVE enables it. For more information, refer to: https://docs.aave.com/developers/v/2.0/the-core-protocol/lendingpool#deposit.

Status

Resolved

21. lpVault is redundant across all Curve contracts

`lpVault` is being set across all Curve contracts but is never used.

Recommendation

Consider removing all occurrences of `lpVault` in order to save on deployment gas fees.

Status

Resolved

22. SigningManager.sol: Function getDomainSeparator() unnecessarily uses assembly to fetch the current chain id

Recommendation

Consider using `block.chainid` instead.

Status

Resolved

23. SharesRegistry.sol: Oracle might return 0

The `SharesRegistry` contract retrieves the current exchange rate from an oracle in `updateExchangeRate()` and caches the value in the storage variable `exchangeRate`, but there is no check if the returned exchange rate is valid (`> 0`).

Recommendation

Validate the returned oracle exchange `rate` and revert if `rate = 0`.

Status

Resolved

24. PandoraUSD.sol: Use ERC20.totalSupply instead of totalMinted

`PandoraUSD.sol` uses a custom storage variable `totalMinted` to keep track of the total amount of minted PandoraUSD tokens. This variable is not necessary and can be easily replaced with `totalSupply` from the ERC20 standard.

Recommendation

Consider using `totalSupply` from `ERC20` and remove the storage variable `totalMinted`.

Status

Resolved

25. Contracts inheriting `Ownable` have `renounceOwnership()` functionality

Renouncing ownership will leave the contract without an owner. This functionality is desirable in certain scenarios and is typically allowed by libraries such as `Ownable`. Following contracts inherit from `Ownable`, thus having the functionality to renounce ownership:

- `HoldingManager`
- `Manager`
- `ManagerContainer`
- `PandoraMerkle`
- `StablesManager`
- `Staker`
- `StrategyManager`
- `SigningManager`
- `ModChainlinkOracle`
- `StrategyBase`
- `ConvexBase`

- `CurveBase`
- `Curve3PoolLPGetter`
- `CurveDaiUsdcUsdtLPGetter`
- `CurveTricryptoLPGetter`

Recommendation

Consider overriding `renounceOwnership()` function in contracts to prevent renouncing ownership accidentally.

Status

Resolved

26. OperationsLib.sol: Use .selector instead of a hexadecimal number

---

In the contract `OperationsLib.sol` the function selector in `safeApprove()` is encoded as a hexadecimal number. Solidity has the keyword `.selector` which makes the code easier to read and less error prone.

Recommendation

Consider using `bytes4(keccak256("approve(address,uint256)"))` instead of a hexadecimal number.

Status

Resolved

27. Gas Optimizations

---

1. `HoldingManager.sol`: Only transfer reward for creating new holding if `mintingTokenReward > 0` in `createHolding()`. Resolved
2. `HoldingManager.sol`: Only transfer first deposit amount when self-assigning holding if `firstDepositAmount > 0` in `assignHoldingToMyself()`. Resolved
3. `Holding.sol`: `require(_balance >= _amount, "2001");` check in `transfer()` is not needed as this is already checked in the OpenZeppelin ERC20 implementation. Resolved
4. `StrategyManager.sol`: `require(strategyInfo[_strategy].whitelisted, "3060");` check in `updateStrategy()` is not needed as this is already checked in the `validStrategy` modifier. Resolved
5. `StablesManager.sol#721`: Cache recurring storage access to `totals[IERC20(_token)]` in variable
6. `StablesManager.sol#751`: Cache recurring storage access to `totals[IERC20(_token)]` in variable
7. `Curve3Pool.sol#122`: Use cached variable `depositToken` instead of reading `tokenIn` from storage. Resolved
8. `Curve3Pool.sol#138`: Use cached variable `depositToken` instead of reading `tokenIn` from storage. Resolved
9. `CurveDaiUsdcUsdtPool.sol#122`: Use cached variable `depositToken` instead of reading `tokenIn` from storage. Resolved
10. `CurveDaiUsdcUsdtPool.sol#138`: Use cached variable `depositToken` instead of reading `tokenIn` from storage. Resolved
11. `SharesRegistry.sol`: Unused `RebaseLib` library. Resolved
12. `Curve3PoolBase.sol`: Unused function `_claimRewards`
13. `CurveDaiUsdcUsdtPoolBase.sol`: Unused function `_claimRewards`

14. `SharesRegistry.sol`: Redundant underflow check in `unregisterCollateral()`.

Resolved

15. `HoldingManager.sol`: Modifier `onlyHoldingUser` is redundant. Resolved

28. Misc Notes

---

1. Add `address(0)` validations throughout the contracts to prevent any accidental transfers. Findings:

- `Manager.constructor()` - `_usdc`
- `ManagerContainer.constructor()` - `_manager`
- `AaveStablecoin.constructor()` - `tokenIn`, `tokenOut`

Resolved

2. `StablesManager.sol#22`: Update NatSpec comment and remove `AlcBox`. Resolved

3. `SigningManager.sol`: Remove references of `AlcBox`. Resolved

4. Duplicate NatSpec function parameter documentation. Findings:

- `./strategies/curve/3pool/Curve3Pool.sol:146` - param `_asset`
- `./strategies/curve/tricrypto/CurveTricrypto.sol:143` - param `_asset`
- `./strategies/curve/daiUsdcUsdt/CurveDaiUsdcUsdtPool.sol:146` - param `_asset`
- `./strategies/convex/3pool/Convex3Pool.sol:157` - param `_asset`
- `./strategies/convex/daiUsdcUsdt/ConvexDaiUsdcUsdt.sol:152` - param `_asset`

Resolved

5. Events are not indexed. Findings:

- `TwapUniswapV2Oracle.sol` - event `PriceUpdated`
- `IMerkleDistributor.sol` - event `Claimed`
- `IModChainlinkOracle.sol` - event `AggregatorAdded`

- `ISharesRegistry.sol` - event `BorrowedSet`
- `ISharesRegistry.sol` - event `BorrowedSharesSet`

Resolved

6. `CurveDaiUsdcUsdtLPGetter.sol#13`: Wrong mentioning of `Convex`. Replace with `Curve`.

Resolved

7. Function visibility should be `internal`. Findings:

- `YearnOracle.sol` - function `_get`
- `CurveOracle.sol` - function `_get`
- `ChainlinkOracle.sol` - function `_get`
- `ModChainlinkOracle.sol` - function `_get`
- `FairUniswapV2Oracle.sol` - function `_get`
- `SimpleUniswapV2Oracle.sol` - function `_get`

Resolved

8. `StablesManager.sol`: Consider using `accrue()` to DRY. Findings:

- `StablesManager.sol#372`
- `StablesManager.sol#448`

Resolved

9. `PandoraUSD.sol#37`: Too many zeros used to represent the number `1000000` for `mintLimit`, consider using scientific notation `1e6` and use the constant `DECIMALS`; `mintLimit = 1e6 * (10**DECIMALS)`. Resolved

10. `PandoraUSD.sol#43`: Wrong NatSpec function comment for `updateMintLimit()`.

Resolved

11. `TwapUniswapV2Oracle.sol`: The NatSpec function comment for `peek()` is mentioning the wrong amount of decimals. It should be `1e18` decimals. Resolved

12. `ModChainlinkOracle.sol`: Consider moving `require` statements in `addAggregator()` to `_addAggregator()`. Resolved

13. `SharesRegistry.sol`: Consider adding a setter function for `oracle`. Resolved

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Pandora 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