SOLIDIFIED

Audit Report for Persistence - August 2, 2021

Summary

Audit Report prepared by Solidified covering the pStake Ethereum smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief on 2 August 2021.

Audited Files

The source code has been supplied in the form of a GitHub repository:

https://github.com/persistenceOne/pStake-smartContracts

Commit number: `b11f4c83614facb5554d26ec6e39545f089e82e1`

The scope of the audit was limited to the following files:

```
contracts
├── LiquidStaking.sol
├── Migrations.sol
├── PSTAKE.sol
├── STokens.sol
├── StakeLPCore.sol
├── TokenWrapper.sol
├── UTokens.sol
├── holder
│   └── HolderUniswap.sol
├── interfaces
│   ├── IHolder.sol
│   ├── ILiquidStaking.sol
│   ├── IPSTAKE.sol
│   ├── ISTokens.sol
│   ├── IStakeLPCore.sol
│   ├── ITokenWrapper.sol
│   └── IUTokens.sol
└── libraries
    ├── Bech32.sol
    ├── Bech32Validation.sol
    ├── BytesLib.sol
    ├── FullMath.sol
    └── TransferHelper.sol
```

Intended Behavior

The smart contracts audited in this phase of the protocol's development implement the tokens and staking primitives necessary for liquid staking.

Code Complexity and Test Coverage

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 equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

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

Issues Found

Solidified found that the pStake contracts contain no critical issue, no major issue, 5 minor issues in addition to 6 informational notes.

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

Issue #	Description	Severity	Status
1	TokenWrapper.sol: Allows fee that is more than 100%	Minor	Pending
2	StakeLPCore.sol: The function removeLiquidity() does not follow the Checks-Effects-Interactions pattern	Minor	Pending
3	TokenWrapper.sol: Event emitted only for the last token	Minor	Pending
4	StakeLPCore.sol: Missing emit keyword for events	Minor	Pending
5	HolderUniswap.sol: Wrong variable updated	Minor	Pending
6	Wide compiler version pragma will not work with all versions	Note	-
7	STokens.sol and UTokens.sol: Potential restriction with integrations with other DeFi products	Note	-
8	STokens.sol: Function isContractWhitelisted() - an extra mapping could be used to avoid the search loop	Note	-
9	HolderUniswap.sol: Unnecessary parameters	Note	-
10	The values stored in the smart contract are not private	Note	-
11	Code cleanup	Note	-

Critical Issues

No critical Issues found.

Major Issues

No major Issues found.

Minor Issues

1. TokenWrapper.sol: Allows fee that is more than 100%

The function `setFees()` contains a `require` statement that ensures the deposit fee and withdrawal fee are less than 100% of the amount. But the `require` statement is missing brackets while validating multiple AND/OR which results in wrong conditions. This allows the admin to set values that are more than 100% of the amount.

Recommendation
Consider grouping the validations to yield expected results.

2. StakeLPCore.sol: The function removeLiquidity() does not follow the Checks-Effects-Interactions pattern

The function `removeLiquidity()` reduces the user's balance after transferring the `lpToken` tokens to the user.

Recommendation
Consider updating user balance before transferring the tokens to minimize the possibility of a re-entrancy based attack.

3. TokenWrapper.sol: Event emitted only for the last token

The method `generateUTokensInBatch` emits the `GenerateUTokens` event only for the last generated token.

Recommendation
Consider emitting the event for all tokens if it was not intended.

4. StakeLPCore.sol: Missing emit keyword for events

Some events do not use the `emit` keyword while emitting an event. By default, the compiler prevents the code from compiling, but there is a bug in some solidity compiler versions which allows events without the `emit` keyword.

Recommendation
We recommend adding the `emit` keyword to all the events emitted to avoid compatibility issues.

5. HolderUniswap.sol: Wrong variable updated

The contract defines two different variables `_sTokenContract` and `_sToksn` to store the `ISTokens` address. Considering how the variables and update methods are used across the project, it looks like this was by mistake and the developer intended to use one variable name.

Recommendation
Consider using the same variable name for `ISTokens` contract.

Informational Notes

6. Wide compiler version pragma will not work with all versions

The codebase specifies compiler version `0.7.0` or above to be used. However, this is a very wide range and will not work in all cases.
For example, `FullMath.sol` library only compiles with versions below `0.8.0`.
Furthermore, the `BytesLib.sol` library used in the project was specifically written for compiler equal or above version `0.8.0`, since it has math overflow checks removed - see the commits history of https://github.com/GNSPS/solidity-bytes-utils/commits/feat/update-0.8.0/contracts/BytesLib.sol .

Recommendation
Consider specifying an exact Solidity compiler version.

7. STokens.sol and UTokens.sol: Potential restriction with integrations with other DeFi products

The functions `burn()` and `mint()` require that the target account of `mint/burn` operation equals `tx.origin`.
While this strict requirement reduces the attack surface it also prevents from `minting/burning` tokens `to/from` contract addresses (only Externally Owned Accounts which can sign an Ethereum transaction are allowed). Such a restriction might prevent future integrations with other DeFi products.

Recommendation
Consider if restricting `tx.origin` is really necessary.

8. STokens.sol: Function isContractWhitelisted() - an extra mapping could be used to avoid the search loop

The function `isContractWhitelisted()` could utilize a mapping from `lpContractAddress` to an `index` of its `_whitelistedAddresses` to eliminate the loop.

Recommendation
Consider if it is worth creating such an extra mapping.

9. HolderUniswap.sol: Unnecessary parameters

The function `getTokenSupply()` declares the parameters `from` and `amount`, which are never used and seem unnecessary in a supply function.

Recommendation
Review if these parameters are necessary in any scenario for this interface.

10. The values stored in the smart contract are not private

The contracts define several private variables. From the comments provided, it looks like the contract is trying to hide these values from the user/public.

Blockchains are generally designed to be transparent and any value stored in the Ethereum smart contract can be easily retrieved. The private keyword makes the values to be hidden from the contract interface, but it can be read from the blockchain directly.

Recommendation
This note is added to make sure the contract developer and user understands how the private methods work.

11. Code cleanup

Consider cleaning up the code based on the following recommendations.

1. `StakeLPCore.sol`: function `calculateRewardsAndLiquidity()` is never used.
2. `StakeLPCore.sol`: missing curly brackets for branches of `if` statement in function `_calculateRewardsAndLiquidity()` at lines 105 and 109
3. `STokens.sol`: function `setWhitelistedAddress()` conducts an unnecessary check: `if (!_whitelistedAddresses.contains(whitelistedAddress))`
4. `STokens.sol`: Incorrect documentation and incorrect comments in functions `isContractWhitelisted()` and `getHolderData()`
5. `TokenWrapper.sol`: Unused import `libraries/Bech32Validation.sol`
6. Library `Bech32Validation.sol` is not used

Disclaimer

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

Solidified Technologies Inc.