pSTAKE Finance

Date	August 2021
Auditors	Shayan Eskandari, Sergii Kravchenko, Eli Leers

1 Executive Summary

This report presents the results of our engagement with **pSTAKE Finance** to review **pSTAKE liquid staking solution**.

The review was conducted over two weeks, from **August 16, 2021** to **August 27, 2021**. A total of 6 person-weeks were spent.

This report includes comments from the pSTAKE Finance team regarding resolutions to the issues originally reported. These resolutions have not been verified by the Consensys Diligence team.

2 Scope

Our review focused on the commit hash `ca3e035454dc565a762a1cc03fdd99c7bf52da37`. The list of files in scope can be found in the Appendix. Note that the bridge functionality, and any technical details regarding Cosmos blockchain are outside the scope of this audit.

3 Recommendations

3.1 Simplify Code flow

Description

Complex code reduces readability and increases the chances of undetected bugs in the system.

Examples

Here are a list of some examples that the code can be significantly simplified:

code/contracts/STokensV2.sol:L216-L292

```
function _calculatePendingRewards(
 uint256 principal,
 uint256 lastRewardTimestamp
```

```
) internal view returns (uint256 pendingRewards) {
 uint256 _index;
 uint256 _rewardBlocks;
 uint256 _simpleInterestOfInterval;
 uint256 _temp;
 // return 0 if principal or timeperiod is zero
 if (principal == 0 || block.timestamp.sub(lastRewardTimestamp) == 0)
  return 0;
 // calculate rewards for each interval period between rewardRate changes
 uint256 _lastMovingRewardLength = _lastMovingRewardTimestamp.length.sub(
  1
 );
 for (_index = _lastMovingRewardLength; _index >= 0; ) {
  // logic applies for all indexes of array except last index
  if (_index < _lastMovingRewardTimestamp.length.sub(1)) {
   if (_lastMovingRewardTimestamp[_index] > lastRewardTimestamp) {
    _rewardBlocks = (_lastMovingRewardTimestamp[_index.add(1)])
     .sub(_lastMovingRewardTimestamp[_index]);
    _temp = principal.mulDiv(_rewardRate[_index], 100);
    _simpleInterestOfInterval = _temp.mulDiv(
     _rewardBlocks,
     _valueDivisor
    );
    pendingRewards = pendingRewards.add(
     _simpleInterestOfInterval
    );
   } else {
    _rewardBlocks = (_lastMovingRewardTimestamp[_index.add(1)])
     .sub(lastRewardTimestamp);
    _temp = principal.mulDiv(_rewardRate[_index], 100);
    _simpleInterestOfInterval = _temp.mulDiv(
     _rewardBlocks,
     _valueDivisor
    );
    pendingRewards = pendingRewards.add(
     _simpleInterestOfInterval
    );
    break;
   }
  }
  // logic applies only for the last index of array
  else {
   if (_lastMovingRewardTimestamp[_index] > lastRewardTimestamp) {
    _rewardBlocks = (block.timestamp).sub(
     _lastMovingRewardTimestamp[_index]
    );
    _temp = principal.mulDiv(_rewardRate[_index], 100);
    _simpleInterestOfInterval = _temp.mulDiv(
     _rewardBlocks,
     _valueDivisor
    );
    pendingRewards = pendingRewards.add(
     _simpleInterestOfInterval
    );
   } else {
    _rewardBlocks = (block.timestamp).sub(lastRewardTimestamp);
    _temp = principal.mulDiv(_rewardRate[_index], 100);
    _simpleInterestOfInterval = _temp.mulDiv(
     _rewardBlocks,
     _valueDivisor
    );
    pendingRewards = pendingRewards.add(
```

```
      _simpleInterestOfInterval
     );
     break;
    }
   }

   if (_index == 0) break;
   else {
    _index = _index.sub(1);
   }
  }
  return pendingRewards;
```

- Decrement `_index` in the for loop statement, which removes the final if statement.

```
    for (  uint256 _index = _lastMovingRewardLength;
     _index >= 0;
     _index = _index.sub(1))
   ...

    // if (_index == 0) break;
    // else {
    //  _index = _index.sub(1);
    // }
```

code/contracts/STokensV2.sol:L531-L558

```
function removeWhitelistedAddress(address whitelistedAddress)
 public
 virtual
 returns (bool success)
{
 require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), "ST10");
 require(whitelistedAddress != address(0), "ST11");
 // remove whitelistedAddress from the list
 _whitelistedAddresses.remove(whitelistedAddress);
 address _holderContractAddressLocal = _holderContractAddress[
  whitelistedAddress
 ];
 address _lpContractAddressLocal = _lpContractAddress[
  whitelistedAddress
 ];

 // delete holder contract values
 delete _holderContractAddress[whitelistedAddress];
 delete _lpContractAddress[whitelistedAddress];

 emit RemoveWhitelistedAddress(
  whitelistedAddress,
  _holderContractAddressLocal,
  _lpContractAddressLocal,
  block.timestamp
 );
 success = true;
 return success;
```

- Unnecessary Complexity, which can be reduced to something like the following:

```
function removeWhitelistedAddress(address whitelistedAddress)
 public
 virtual
 returns (bool success)
{
 require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), "ST10");
 require(whitelistedAddress != address(0), "ST11"); // not really needed for this function
 // remove whitelistedAddress from the list
 _whitelistedAddresses.remove(whitelistedAddress);

 emit RemoveWhitelistedAddress(
  whitelistedAddress,
  _holderContractAddress[whitelistedAddress],
  _lpContractAddress[whitelistedAddress],
  block.timestamp
 );

 // delete holder contract values
 delete _holderContractAddress[whitelistedAddress];
 delete _lpContractAddress[whitelistedAddress];

 return true;
}
```

- The `_beforeTokenTransfer()` function in STokensV2 has too many if conditions, it is possible to reduce the number of these conditions.

3.2 Track fees properly

Description

Currently on staking and unstaking the fees are calculated but the values are not stored. The way it works is that the fees are reduced from the amount sent to user and are left in the contract, which using events such as `StakeTokens(to, amount, _finalTokens, block.timestamp);` fees are calculated off-chain (finalTokens - amount).

Recommendation

It is suggested to track the fees on-chain, as it will be easier to implement proper unit tests and identify any discrepancies in the test environment.

4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

4.1 Actors

The relevant actors are listed below with their respective abilities:

- Admin
  - Currently an EOA controlled by pStake team (same entity as deployer), eventually will be a multisig account

    - Can upgrade the contracts
    - Can change the Token contracts, Staking and LP contract, and wrapper contract
    - Can set change fees, reward rates
    - Can set and remove whitelisted addresses
    - Can set the unstaking epoch, lock time and minimum values
    - Can change all other admin addresses through OZ ACL
- Pauser (`pauserAddress`)
    - Can pause and unpause the system
- Bridge Admin
    - Can mint & burn uTokens
- pToken Holders (uToken)
    - Can deposit pTokens to receive sTokens (Stake)
    - Can withdraw pTokens and receive ATOM (minus fees)
- sToken Holders
    - Can unstake sTokens to receive pTokens
    - Receives rewards for their sTokens
    - Stake sTokens in DeFi Pools and stake the LP tokens to earn rewards & yield
- Anyone
    - Can deposit ATOM to receive pTokens

4.2 Trust Model

In any system, it's important to identify what trust is expected/required between various actors. For this audit, we established the following trust model:

- Users are trusting pStake team `admin` as they have the capability to upgrade any contract in this system at will. Also many variables in the system, such as fees, valid DeFi pools, unstaking epoch, etc, can be changed by the owner.
- The bridge is responsible to mint new tokens, and it is crucial to verify the token flow cross-chains. Note that the bridge implementation is not part of this audit.
- Currently the initial total supply of PSTAKE tokens are minted to the deployer address with the goal to be airdropped. The pStake team mentioned that they have a vesting mechanism in the road map.

5 Findings

Each issue has an assigned severity:

- `Minor` issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.

- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 Reward rate changes are not taken into account in LP staking

Major

Resolution
Comment from pSTAKE Finance team: > Have implemented two new Emission logic for pToken & Other Reward Tokens in StakeLP which mandatorily distributes rewards only after updating the Reward Pool, thereby fixing this potential issue.

Description

When users update their reward (e.g., by calling the `calculateRewards` function), the reward amount is calculated according to all reward rate changes after the last update. So it does not matter when and how frequently you update the reward; in the end, you're going to have the same amount.

On the other hand, we can't say the same about the lp staking provided in the `StakeLPCoreV8` contract. The amount of these rewards depends on when you call the `calculateRewardsAndLiquidity` function, and the reward amount can even decrease over time.

Two main factors lead to this:

- Changes in the reward rate. If the reward rate is decreased at some point, it's getting partially propagated to all the rewards there were not distributed yet. So the reward of the users that didn't call the `calculateRewardsAndLiquidity` function may decrease. On the other hand, if the reward rate is supposed to increase, it's better to wait and not call `calculateRewardsAndLiquidity` for as long as possible.
- Not every liquidity provider will stake their LP tokens. When users provide liquidity but do not stake the LP tokens, the reward for these Stokens is still going to the Holder contract. These rewards getting proportionally distributed to the users that are staking their LP tokens. Basically, these rewards are added to the current reward rate but change more frequently. The same logic applies to that rewards; if you expect the unstaked LP tokens to increase, it's in your interest not to withdraw your rewards. But if they are decreasing, it's better to gather the rewards as early as possible.

Recommendation

The most preferred staking solution is to have an algorithm that is not giving people an

incentive to gather the rewards earlier or later.

5.2 The `withdrawUnstakedTokens` may run out of gas Major

Resolution
Comment from pSTAKE Finance team: > Have implemented a batchingLimit variable which enforces a definite number of iterations during withdrawal of unstaked tokens, instead of indefinite iterations.

Description

The `withdrawUnstakedTokens` is iterating over all batches of unstaked tokens. One user, if unstaked many times, could get their tokens stuck in the contract.

code/contracts/LiquidStakingV2.sol:L369-L403

```
function withdrawUnstakedTokens(address staker)
 public
 virtual
 override
 whenNotPaused
{
 require(staker == _msgSender(), "LQ20");
 uint256 _withdrawBalance;
 uint256 _unstakingExpirationLength = _unstakingExpiration[staker]
  .length;
 uint256 _counter = _withdrawCounters[staker];
 for (
  uint256 i = _counter;
  i < _unstakingExpirationLength;
  i = i.add(1)
 ) {
  //get getUnstakeTime and compare it with current timestamp to check if 21 days + epoch difference ha
  (uint256 _getUnstakeTime, , ) = getUnstakeTime(
   _unstakingExpiration[staker][i]
  );
  if (block.timestamp >= _getUnstakeTime) {
   //if 21 days + epoch difference has passed, then add the balance and then mint uTokens
   _withdrawBalance = _withdrawBalance.add(
    _unstakingAmount[staker][i]
   );
   _unstakingExpiration[staker][i] = 0;
   _unstakingAmount[staker][i] = 0;
   _withdrawCounters[staker] = _withdrawCounters[staker].add(1);
  }
 }

 require(_withdrawBalance > 0, "LQ21");
 emit WithdrawUnstakeTokens(staker, _withdrawBalance, block.timestamp);
 _uTokens.mint(staker, _withdrawBalance);
}
```

Recommendation

Limit the number of processed unstaked batches, and possibly add pagination.

5.3 The `_calculatePendingRewards` can run out of gas Medium

Resolution
Comment from pSTAKE Finance team: A solution of maintaining cumulative timeshare values in an array and implementing binary search drastically lowers the iterations, has been implemented for calculating rewards for Other Reward Tokens. Also, for moving reward rate, strategically it will only be set max once a month making number of iterations very limited.

Description

The reward rate in STokens can be changed, and the history of these changes are stored in the contract:

code/contracts/STokensV2.sol:L124-L139

```
function setRewardRate(uint256 rewardRate)
 public
 virtual
 override
 returns (bool success)
{
 // range checks for rewardRate. Since rewardRate cannot be more than 100%, the max cap
 // is _valueDivisor * 100, which then brings the fees to 100 (percentage)
 require(rewardRate <= _valueDivisor.mul(100), "ST17");
 require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), "ST2");
 _rewardRate.push(rewardRate);
 _lastMovingRewardTimestamp.push(block.timestamp);
 emit SetRewardRate(rewardRate);

 return true;
}
```

When the reward is calculated for each user, all changes of the `_rewardRate` are considered. So there is a `for` loop that iterates over all changes since the last reward update. If the reward rate was changed many times, the `_calculatePendingRewards` function could run out of gas.

Recommendation

Provide an option to partially update the reward, so the full update can be split in multiple transactions.

5.4 Increase test coverage Medium

Resolution
Comment from pSTAKE Finance team: Created deep-dive test for each unique scenarios locally and tested before the code was deployed.

Description

Test coverage is fairly limited. LPStaking tests only cover the happy path. StakeLPCoreV8 has no tests. Many test descriptions are inaccurate.

Examples

Test description inaccuracy examples:

- This tests that a Staker can mint new tokens, but does not check to make sure that Stakers are the ONLY group that can mint. https://github.com/ConsenSys/persistence-pstake-audit-2021-08/blob/3821182ca14e0e98ab9fccd47cbe0f1ce39ae54c/code/test/LiquidStakingTest.js#L82
- This test only shows that an unauthorized address can't use the stake function to mint tokens. https://github.com/ConsenSys/persistence-pstake-audit-2021-08/blob/3821182ca14e0e98ab9fccd47cbe0f1ce39ae54c/code/test/LiquidStakingTest.js#L99
- This test actually tests for the inverse case. https://github.com/ConsenSys/persistence-pstake-audit-2021-08/blob/3821182ca14e0e98ab9fccd47cbe0f1ce39ae54c/code/test/STokensTest.js#L82

Recommendation

Increase test coverage for entire codebase. Add tests for the inherited contracts from OpenZeppelin. Test for edge cases, and multiple expected cases. Ensure that the test description matches the functionality that is actually tested.

5.5 The `calculateRewards` should not be callable by the whitelisted contract Medium

Resolution
Comment from pSTAKE Finance team: Have created a require condition in Smart Contract code to disallow whitelisted contracts from calling the function

Description

The `calculateRewards` function should only be called for non-whitelisted addresses:

code/contracts/STokensV2.sol:L348-L359

```
function calculateRewards(address to)
 public
 virtual
 override
 whenNotPaused
 returns (bool success)
{
 require(to == _msgSender(), "ST5");
 uint256 reward = _calculateRewards(to);
 emit TriggeredCalculateRewards(to, reward, block.timestamp);
 return true;
}
```

For all the whitelisted addresses, the `calculateHolderRewards` function is called. But if the

`calculateRewards` function is called by the whitelisted address directly, the function will execute, and the rewards will be distributed to the caller instead of the intended recipients.

Recommendation

While this scenario is unlikely to happen, adding the additional check in the `calculateRewards` is a good option.

5.6 Presence of testnet code Medium

Resolution
Comment from pSTAKE Finance team: The testnet code has been re-considered as out of scope for audit

Description

Based on the discussions with pStake team and in-line comments, there are a few instances of code and commented code in the code base under audit that are not finalized for mainnet deployment.

Examples

code/contracts/PSTAKE.sol:L25-L37

```
function initialize(address pauserAddress) public virtual initializer {
 __ERC20_init("pSTAKE Token", "PSTAKE");
 __AccessControl_init();
 __Pausable_init();
 _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
 _setupRole(PAUSER_ROLE, pauserAddress);
 // PSTAKE IS A SIMPLE ERC20 TOKEN HENCE 18 DECIMAL PLACES
 _setupDecimals(18);
 // pre-allocate some tokens to an admin address which will air drop PSTAKE tokens
 // to each of holder contracts. This is only for testnet purpose. in Mainnet, we
 // will use a vesting contract to allocate tokens to admin in a certain schedule
 _mint(_msgSender(), 5000000000000000000000000);
}
```

The initialize function currently mints all the tokens to msg.sender, however the goal for mainnet is to use a vesting contract which is not present in the current code.

Recommendation

It is recommended to fully test the **final** code before deployment to the mainnet.

5.7 Re-entrancy from LP token transfers Minor

Resolution

Comment from pSTAKE Finance team:

> Have implemented the nonReentrant modifier from the ReentrancyGuardUpgradeable OpenZeppelin contract in addition to strictly keeping to Checks-Effects-Interactions pattern throughout relevant areas

Description

The `StakeLPCoreV8` contract is designed to stake LP tokens. These LP tokens are not directly controlled or developed by the protocol, so it can't be easily verified that no re-entrancy can happen during token transfers.

Recommendation

During the review, we did not find any specific ways to build the attack using the re-entrancy of LP tokens, but it is still better to have the re-entrancy protection modifiers in functions that use LP tokens transfers.

5.8 Sanity check on all important variables Minor

Resolution

Comment from pSTAKE Finance team:

> Post the implementation of new emission logic there have been a rearrangement of some variables, but the rest have been sanity tested and corrected

Description

Most of the functionalities have proper sanity checks when it comes to setting system-wide variables, such as whitelist addresses. However there are a few key setters that lack such sanity checks.

Examples

- Sanity check (!= address(0)) on all token contracts.

code/contracts/StakeLPCoreV8.sol:L303-L333

```
function setUTokensContract(address uAddress) public virtual override {
 require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), "LP9");
 _uTokens = IUTokens(uAddress);
 emit SetUTokensContract(uAddress);
}

/**
 * @dev Set 'contract address', called from constructor
 * @param sAddress: stoken contract address
 *
 * Emits a {SetSTokensContract} event with '_contract' set to the stoken contract address.
 *
 */
function setSTokensContract(address sAddress) public virtual override {
 require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), "LP10");
 _sTokens = ISTokens(sAddress);
 emit SetSTokensContract(sAddress);
}

/**
 * @dev Set 'contract address', called from constructor
 * @param pstakeAddress: pStake contract address
 *
 * Emits a {SetPSTAKEContract} event with '_contract' set to the stoken contract address.
 *
 */
function setPSTAKEContract(address pstakeAddress) public virtual override {
 require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), "LP11");
 _pstakeTokens = IPSTAKE(pstakeAddress);
 emit SetPSTAKEContract(pstakeAddress);
}
```

- Sanity check on `unstakingLockTime` to be in the acceptable range (21 hours to 21 days)

code/contracts/LiquidStakingV2.sol:L105-L121

```
/**
 * @dev Set 'unstake props', called from admin
 * @param unstakingLockTime: varies from 21 hours to 21 days
 *
 * Emits a {SetUnstakeProps} event with 'fee' set to the stake and unstake.
 *
 */
function setUnstakingLockTime(uint256 unstakingLockTime)
 public
 virtual
 returns (bool success)
{
 require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), "LQ3");
 _unstakingLockTime = unstakingLockTime;
 emit SetUnstakingLockTime(unstakingLockTime);
 return true;
}
```

5.9 Remove unused/commented code

Description

There are a few snippets of commented code in the code base. It is suggested to remove and clean any unused and commented code in the final code.

Examples

code/contracts/PSTAKE.sol:L50-L73

```
/* function mint(address to, uint256 tokens) public virtual override returns (bool success) {
       require(_msgSender() == _stakeLPCoreContract, "PS1");  // minted by STokens contract

       _mint(to, tokens);
       return true;
   } */

/*
 * @dev Burn utokens for the provided 'address' and 'amount'
 * @param from: account address, tokens: number of tokens
 *
 * Emits a {BurnTokens} event with 'from' set to address and 'tokens' set to amount of tokens.
 *
 * Requirements:
 *
 * - `amount` cannot be less than zero.
 *
 */
/* function burn(address from, uint256 tokens) public virtual override returns (bool success) {
       require((tx.origin == from && _msgSender()==_liquidStakingContract) ||  // staking operation
       (tx.origin == from && _msgSender() == _wrapperContract), "UT2"); // unwrap operation
       _burn(from, tokens);
       return true;
   } */
```

Appendix 1 - Files in Scope

This audit covered the following files:

File Name	SHA-1 Hash
contracts/PSTAKE.sol	c05ec40a3c51e6944108bd51e01a91675f20ba21
contracts/StakeLPCoreV8.sol	5096dc6a5fa35ce8b5d1559e1de9892846ab31ed
contracts/STokensV2.sol	c0cdc3539a5791e0cc85d93ac59369e2429eeec4
contracts/TokenWrapperV2.sol	3309ca01751f1bf3db0905e0f4e40e4011ee364e
contracts/LiquidStakingV2.sol	42f7b9c5aa895e99da8740b8017f89a045e87e3a
contracts/UtokensV2.sol	ab0ccc98d5cc0033a6cfbc5c747e06dfbb0906b1
contracts/holder/HolderUniswapV2.sol	9df41bf94932473741e82d9c3d7a4bd8d460e383
contracts/interfaces/IStakeLPCore.sol	907d06932218dcfcd44aa0139818fb04f35f014d
contracts/interfaces/ILiquidStaking.sol	c86123c802153803d8dd036fd0a73643b3e14b23
contracts/interfaces/IPSTAKE.sol	ddc9371228ced1468e9f6be388271e63046b7cf3
contracts/interfaces/IHolder.sol	ea375ca85083fb33a5db3dbec2b00c6ebf811009
contracts/interfaces/ITokenWrapper.sol	c7199aa8b27e59867174ae8c9bb3465f9d475a9a
contracts/interfaces/ISTokens.sol	af3816acce8d54c9de2df44d72a6f586b94f98ac
contracts/interfaces/IUTokens.sol	75bdab31606a0782262e9cb136b589cd1c6f4d05
contracts/libraries/BytesLib.sol	b994356e1c3d5eece267c342f768470bfbceac94
contracts/libraries/FullMath.sol	183001f929ce4003224e9eae67b4b6e12adac3b2
contracts/libraries/TransferHelper.sol	8ecaf833f0fc0b9e8c786fd65fa51ef16094ec3e
contracts/libraries/Bech32.sol	375f3d9a77d02963e9dc274eea4b79679ef4f7ea

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is

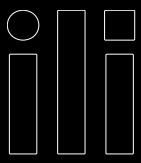
not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) - on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.



Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.