Keep3rV2 Audit Report

PeckShield

Oct 10, 2021

PVE001: Improved Logic of Keep3rKeeperFundable::bond()

```
/**
 * @notice begin the bonding process for a new keeper
 * @param _bonding the asset being bound
 * @param _amount the amount of bonding asset being bound
 */
function bond(address _bonding, uint256 _amount) external override nonReentrant {
  if (disputes[msg.sender]) revert Disputed();
  if (_jobs.contains(msg.sender)) revert AlreadyAJob();
  canActivateAfter[msg.sender][_bonding] = block.timestamp + bondTime;
  ← if (_keepers.contains(msg.sender)) revert AlreadyAKeeper();
  uint256 _before = IERC20(_bonding).balanceOf(address(this));
  IERC20(_bonding).safeTransferFrom(msg.sender, address(this), _amount);
  _amount = IERC20(_bonding).balanceOf(address(this)) - _before;

  hasBonded[msg.sender] = true;
  pendingBonds[msg.sender][_bonding] += _amount;
  emit Bonding(msg.sender, block.number, canActivateAfter[msg.sender][_bonding], _amount);
}
```

PVE002: Suggested Adherence Of Checks-Effects-Interactions Pattern

- Suggest to apply state changes before the external interaction

```
 */
function withdraw(address _bonding) external override nonReentrant {
  if (canWithdrawAfter[msg.sender][_bonding] == 0) revert UnbondsUnexistent();
  if (canWithdrawAfter[msg.sender][_bonding] >= block.timestamp) revert UnbondsLocked();
  if (disputes[msg.sender]) revert Disputed();

  uint256 _amount = pendingUnbonds[msg.sender][_bonding];

  if (_bonding == keep3rV1) {
    IKeep3rV1Proxy(keep3rV1Proxy).mint(_amount);
  }

  IERC20(_bonding).safeTransfer(msg.sender, _amount);

  emit Withdrawal(msg.sender, _bonding, _amount);
  pendingUnbonds[msg.sender][_bonding] = 0;
}
```

Interactions

Effects

- Affected
  - Keep3rKeeperFundable::withdraw(), withdrawLiquidityFromJob()

PVE003: Possible FrontRunning DoS Against Job Credit Withdrawal

```
function addTokenCreditsToJob(
  address _token,
  address _job,
  uint256 _amount
) external override nonReentrant {
  if (!_jobs.contains(_job)) revert JobUnavailable();
  // KP3R shouldn't be used for direct token payments
  if (_token == keep3rV1) revert TokenUnavailable();
  uint256 _before = IERC20(_token).balanceOf(address(this));
  IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
  uint256 _received = IERC20(_token).balanceOf(address(this)) - _before;
  uint256 _fee = (_received * FEE) / BASE;
  jobTokenCredits[_job][_token] += _received - _fee;
  jobTokenCreditsAddedAt[_job][_token] = block.timestamp;
  IERC20(_token).safeTransfer(gc
  _jobTokens[_job].add(_token);

  emit AddCredit(_job, _token, m
}
```

A malicious actor can deposit 1 WEI to update the AddedAt timestamp

```
function withdrawTokenCreditsFromJob(
  address _token,
  address _job,
  uint256 _amount,
  address _receiver
) external override nonReentrant onlyJobOwner(_job) {
  if (block.timestamp <= jobTokenCreditsAddedAt[_job][_token] + _WITHDRAW_TOKENS_COOLDOW
  if (jobTokenCredits[_job][_token] < _amount) revert InsufficientJobTokenCredits();

  jobTokenCredits[_job][_token] -= _amount;
  IERC20(_token).safeTransfer(_receiver, _amount);
```

Withdrawal is blocked

PVE004: Simplified Logic of Keep3rJobFundableLiquidity::_phase()

```
/// @notice Returns a fraction of the multiplier or the whole multiplier if equal or more than a rewar
function _phase(uint256 _timePassed, uint256 _multiplier) internal view returns (uint256 _result) {
  if (_timePassed < rewardPeriodTime) {
    _result = ((_timePassed % rewardPeriodTime) * _multiplier) / rewardPeriodTime;
  } else _result = _multiplier;
}
```

_timePassed

PVE005: Improved Quote Calc. of Keep3rLibrary::getQuoteAtTick()

```
/* WARNING Uniswap's getQuoteAtTick was designed for a uint128 baseAmount */
function getQuoteAtTick(
  uint256 baseAmount,
  int56 tickDifference,
  uint256 timeInterval
) public pure returns (uint256 _quoteAmount) {
  uint160 sqrtRatioX96 = getSqrtRatioAtTick(int24(tickDifference / int256(timeInterval)));
  uint256 ratioX128 = mulDiv(sqrtRatioX96, sqrtRatioX96, 1 << 64);
  _quoteAmount = mulDiv(1 << 128, baseAmount, ratioX128);
}
```

Better precision

```
function getQuoteAtTick(
  uint256 baseAmount,
  int56 tickDifference,
  uint256 timeInterval
) public pure returns (uint256 _quoteAmount) {
  uint160 sqrtRatioX96 = getSqrtRatioAtTick(int24(tickDifference / int256(timeInterval)));

  // Calculate quoteAmount with better precision if it doesn't overflow when multiplied by itself
  if (sqrtRatioX96 <= type(uint128).max) {
      uint256 ratioX192 = uint256(sqrtRatioX96) * sqrtRatioX96;
      _quoteAmount = mulDiv(1 << 192, baseAmount, ratioX192);
  } else {
      uint256 ratioX128 = mulDiv(sqrtRatioX96, sqrtRatioX96, 1 << 64);
      _quoteAmount = mulDiv(1 << 128, baseAmount, ratioX128);
  }
}
```

PVE006: Improved Job Migration in Keep3rJobMigration

```
function migrateJob(address _fromJob, address _toJob) external override onlyJobOwner(_fromJob) {
  if (_fromJob == _toJob) revert JobMigrationImpossible();

  pendingJobMigrations[_fromJob] = _toJob;
  _migrationCreatedAt[_fromJob][_toJob] = block.timestamp;

  emit JobMigrationRequested(_from
}
```

```
if (!_jobs.contains(_toJob)) revert JobUnavailable()
```

```
function acceptJobMigration(address _fromJob, address _toJob) external override
  if (disputes[_fromJob] || disputes[_toJob]) revert JobDisputed();
  if (pendingJobMigrations[_fromJob] != _toJob) revert JobMigrationUnavailable()
  if (block.timestamp < _migrationCreatedAt[_fromJob][_toJob] + _MIGRATION_COOLD
  ...
  // migrate job balances
  _jobPeriodCredits[_toJob] += _jobPeriodCredits[_fromJob];
  delete _jobPeriodCredits[_fromJob];

  _jobLiquidityCredits[_toJob] += _jobLiquidityCredits[_fromJob];
  delete _jobLiquidityCredits[_fromJob];

  // stop _fromJob from being active
  delete rewardedAt[_fromJob];
  _jobs.remove(_fromJob);

  pendingJobMigrations[_fromJob] = address(0);
  emit JobMigrationSuccessful(_fromJob, _toJob);
}
```

```
delete jobOwner[_fromJob];
delete jobPendingOwner[_fromJob];
delete _migrationCreatedAt[_fromJob][_toJob
```

PVE007: Possible DoS Against Keep3rJobDisputable::slashJob()

```
 */
function slashJob(address _job) external override nonReentrant onlySlasherOrGovernance {
  if (!disputes[_job]) revert NotDisputed();

  // slash job tokens and token credits
  uint256 _index = 0;
  while (_index < _jobTokens[_job].length()) {
    address _token = _jobTokens[_job].at(_index);

    // make low level call ir order to avoid reverting
    // solhint-disable-next-line avoid-low-level-calls
    try IERC20(_token).transfer(governance, jobTokenCredits[_job][_token]) {
      jobTokenCredits[_job][_token] = 0;
      _jobTokens[_job].remove(_token);
    } catch {
      _index++;
    }
  }
```

A malicious token may be added to cause out-of-gas when being slashed !

- Affected
  - slashTokenFromJob(), slashJob()

PVE008: Improved Logic Of Keep3rJobManager::removeJob()

```
function addJob(address _job) external override {
  if (_jobs.contains(_job)) revert JobAlreadyAdded();
  if (hasBonded[_job]) revert AlreadyAKeeper();
  _jobs.add(_job);
  jobOwner[_job] = msg.sender;
  emit JobAddition(_job, block.number, msg.sender);
}

/**
 * @notice Allows governance to remove a job from the systems
 * @param _job address of the contract for which work should be performed
 */
function removeJob(address _job) external override onlyGovernance {
  if (!_jobs.contains(_job)) revert JobUnexistent();
  _jobs.remove(_job);
  emit JobRemoval(_job, block.number, msg.sender);
}
```

```
delete jobOwner[_fromJob];
delete jobPendingOwner[_fromJob];
```