Smart Contract Source Code Audit Sovryn Governance

Prepared for Sovryn • January 2021

v210115

1. Table Of Contents

2. Executive Summary

In January 2021, Sovryn engaged Coinspect to perform a source code review of their new governance, staking and fee sharing contracts. The objective of the audit was to evaluate the security of the smart contracts implementing these features.

The code reviewed was found to be clear, well written, and properly documented. No high risk vulnerabilities were discovered during this audit.

Even though the new features give the protocol users more participation, it is worth noting, by design, centralized roles are still able to control governance decisions, at least until governance abdicates their proposal veto right.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk	Informational
0	**0**	**1**	**2**

3. Introduction

Sovryn's goal is to enable lending, borrowing and margin trading in the RSK blockchain.

This audit focused on the smart contracts that implement the following recently developed features:

1. Governance
2. Staking
3. Vesting
4. Sovryn Rewards Token

The audit started on January 4th and was conducted on the PR#63 Governance pull request from the Sovryn Git repository. This pull request includes 310 commits and modified 73 files.

This review focused on the following new Solidity source files shown here with their sha256sum hash:

```
5500fd645abe493558849208348b5ca09d5fbcb63862cede9585d075a35ebc4a  ./governance/Vesting/IVesting.sol
6a3cd8228410fb5639c13542d500ac4fc55dd56ccb7fb2a6cecd28c82de2ea87  ./governance/Vesting/Vesting.sol
2616a8c6a5b39046d9f3d64b8b6e83ba24a40cdaffc20ebe63bf445a469285da  ./governance/Vesting/TeamVesting.sol
345ce97154433a1d77bf4d3998080677d0faacd2b6cd037b87ffbe93311fac6b  ./governance/Vesting/RSOV.sol
752d4cb2d6bf3d35dea9084a80facf60935341f803d6f463838d85b0d9d789cd  ./governance/Vesting/DevelopmentVesting.sol
829d54b74fb8eab57f0bddf904300233763fcb593935290bb6d4fc9aa016a35d  ./governance/IFeeSharingProxy.sol
379e1187912e7d676816adb0d264bc963340b376b5bfae339177ea135be36121  ./governance/Staking/StakingStorage.sol
3f3d31dcaa4c0e8766a6d00b5c27b3a738c159cd58bf61e246da7728fce46c54  ./governance/Staking/SafeMath96.sol
3f324cab3adbda475ae5d0c5e8786fdf2cd7309721bd3df550db49d0d66d4cbf  ./governance/Staking/WeightedStaking.sol
3464e5dfdf5f9d31cded3a5dc1e55b69d0948debb41ad275a9806ecceda22b5a  ./governance/Staking/Checkpoints.sol
a4766027aa58e996dd1a99caeb6ccb833da1d8ee5d0fc1cfdafc5629ab0ab0a6  ./governance/Staking/IStaking.sol
d49fd09134e68430f3a1323d4462de352969be164751e271f778c84a174f1efe  ./governance/Staking/StakingProxy.sol
2fad61745acd6d5aad1cf8a27346e9a6522caff4464ce0f80a1baf092f2c5823  ./governance/Staking/Staking.sol
7b03830660d9e887b7526a17e29d83a2226d97a2ca2086580a55f2bb20ac97aa  ./governance/Timelock.sol
3bc9e03898c93961e77de52c9b48acc9e2a71e5a526416212694742522fcec70  ./governance/FeeSharingProxy.sol
bb12a6faf2c9a816b939abc6e671f36f436f5f3403b22d4dab1b07685ac94aff  ./governance/GovernorAlpha.sol
cdcc5deac47f210bcf7032d03dbdb16156f802a6197d71997ad2528590213711  ./proxy/Proxy.sol
```

The changes made to other smart contracts in the platform in order to integrate these new files were reviewed as well.

The following design documentation provided by the Sovryn team was consulted during the assessment:

1. Governance development plan
2. https://hackmd.io/eLCgFHT3QEKre9bf_nyCOA?view#Governance-Brainstorming

4. Assessment

The new governance feature, implemented in the GovernorAlpha.sol and Timelock.sol smart contracts, is based on Compound's governance model implementation. However, the scheme was modified by Sovryn to incorporate weighted voting power based on staking.

The contracts are compiled with Solidity version 0.5.17. A few compile warnings are emitted at build time, most of them related to shadowed variables and the use of ABIEncodeV2, which was experimental for that Solidity release. Several tests were added for the new functionality, all of these 233 new tests included in the repository pass. Coinspect observed the visibility of some public functions could be changed to external in order to optimize gas usage.

The following sections explore each of new features, and provide a brief description and audit notes for each of them.

4.1 Governance

The governance model is based on Compound's system. Executable governance proposals can be made by anybody with voting power above a certain threshold. These proposals are voted, and if approved, queued in the Timelock contract which allows executing them after a period of time.

Votes can be delegated, and are calculated based on the staking voting power taking into account a proposal's start block and start time.

Sovryn's implementation allows the governance guardian role to veto any proposal.

A mechanism for the guardian to abdicate his role is provided.

4.2 Staking

The staking contract allows users to deposit tokens for a period of time, in exchange for voting power and fees sharing rights. This contract allows users to delegate their stake to another account. Also, it is possible to stake with a certain schedule, this functionality is used by the vesting contract.

The weightedStaking.sol contract is responsible for voting power calculations for any point in time. In order to save gas and prevent attacks, this is implemented via a checkpointing mechanism with a 2 weeks period.

It is worth noting, **users are allowed to withdraw their vested tokens before time** with a cost proportional to the time remaining until the original stake finish date. The slashed amount is transferred to the fee sharing contract. The following output from one of the tests provided shows how this punishment mechanism works:

```
Staked amount: 10000
lock date: 2 (weeks), slashed amount: 360 ( 3.6% )
lock date: 20 (weeks), slashed amount: 930 ( 9.3% )
lock date: 40 (weeks), slashed amount: 1500 ( 15% )
lock date: 60 (weeks), slashed amount: 1950 ( 19.5% )
lock date: 80 (weeks), slashed amount: 2340 ( 23.4% )
lock date: 100 (weeks), slashed amount: 2640 ( 26.4% )
lock date: 120 (weeks), slashed amount: 2850 ( 28.5% )
lock date: 140 (weeks), slashed amount: 2970 ( 29.7% )
lock date: 156 (weeks), slashed amount: 3000 ( 30% )
```

As a consequence, 30% of the staked amount gets slashed when the depositor decides to withdraw 156 weeks before the stake period is due, this is the maximum staking period and its punishment.

The Staking contract owner, governance, has the ability to allow any address to withdraw anytime without being punished through a vesting whitelist which is used in the `governanceWithdrawVesting` function.

Additionally, there is an irreversible unlock all tokens flag that can be enabled by the contract owner and disables slashing for all stakes.

A staking proxy contract is also included in the repository, which allows the proxy owner to upgrade the staking contract implementation.

4.3 Vesting

The Vesting contracts enforce two kinds of vesting schedules: one is intended for investor's tokens and the other one for the team's tokens.

Team vesting is implemented in Vesting.sol, the vested tokens get staked in the staking contract with a determined schedule (cliff, duration and period) via the `stakeBySchedule` function. As time passes, vested tokens get unlocked according to the vesting schedule. **The SOV token owner and the Vesting contract owner are the only ones allowed to withdraw tokens**. The withdraw function only allows withdrawing tokens that have been unlocked. There is one exception to this rule: **the Staking contract is allowed to withdraw all tokens anytime, including locked ones, through the `withdrawGovernance` function**.

Investor's vesting is implemented in DevelopmentVesting.sol. In this case, vested tokens do not get staked, the vesting contract just locks the tokens. As a consequence, the investors who deposit tokens in this contract do not get voting or fee sharing rights. Tokens can get deposited or vested in this contract: deposited tokens can be withdrawn anytime, vested tokens are locked in for a period of time. In addition to the contract owner, the token owner is allowed to withdraw tokens from this vesting contract. The withdraw functions can only withdraw unlocked vested tokens or those that were deposited.

The vesting contract owner can unlock all tokens anytime by setting the vesting schedule parameters to zero.

4.4 Rewards and fee sharing

The RSOV token (Sovryn Reward Token) goal is to allow users to get rewards through the generation of protocol fees. The mint function accepts SOV tokens and mints the same amount of RSOV tokens. When burning RSOV tokens, the user gets 1/14th of the tokens sent back to him and the rest get staked in the user's behalf with a schedule of 4 weeks cliff and period and 1 year duration.

The FeeSharingProxy contract is intended to be set as the protocol fee collector. Anybody can invoke the `withdrawFees` function which uses `protocol.withdrawFees` to obtain available fees from operations on a certain token, these fees are deposited in the corresponding `loanPool`. Also, the staking contract sends slashed tokens to this contract. When a user calls the `withdraw` function, the contract transfers the fee sharing rewards in proportion to the user's weighted stake since the last withdrawal.

4.4 Miscellaneous modifications

The following changes were introduced by the same pull request and were reviewed by Coinspect:

1. Added new admin and pauser roles, separate from the contract owner, to the LoanToken contracts.
2. Added new admin role to State.sol and ProtocolSettings.sol smart contracts.
3. Solidity version bump from `^0.4.15` to `^0.5.17` for MultiSigWallet.sol and the required modifications to support the newer version
4. New events were added.
5. `withdrawFees` function added to `ISovryn` protocol interface

5. Conclusions and Recommendations

No high risk vulnerabilities were found during this assessment. The source code reviewed was found to be correct and only minor suggestions are made in this report in order to improve code quality.

Even though the main goal of the new features is to offer the protocol users a more decentralized governance model and more participation in decision making, it is worth noting the system will not be fully decentralized when deployed and there are mechanisms in place, such as the guardian’s veto right, that allow certain roles to maintain control of governance, such as the veto mechanism. This is intended by design, and the mechanisms to forfeit centralized control are in place for when the moment comes for the Sovryn team to switch to a completely decentralized governance model.

Another important design decision that should be taken into account is the users ability to withdraw their stakes, with a penalization, after proposing and/or voting an executable governance proposal.

The following list sums up the most important recommendations from this audit:

1. Review visibility of functions to make sure all functions that can be declared as external are declared that way in order to save gas.
2. Clearly document the roles in the system and their rights to bypass locked tokens withdrawal schedules.
3. Clearly document the ability of the guardian role to veto any governance proposal. The process for the guardian to monitor proposals and decide which ones should be vetoed should be documented.

6. Summary of Findings

ID	Description	Risk	Fixed
SVN-014	Incorrect ecrecover return value checks	Low	✘
SVN-015	Shadowed declarations	Info	✘
SVN-016	Incorrect revert error message strings in Checkpoints.sol	Info	✘

8. Findings

SVN-014 Incorrect ecrecover return value checks

Total Risk	Impact	Location
Low	Low	Staking.sol GovernorAlpha.sol
Fixed	Likelihood	
✘	High	

Description

In the RSK blockchain implementation, `ecrecover`'s precompiled contract return value in error scenarios is the value `0xdcc703c0E500B653Ca82273B7BFAd8045D85a470` in contrast with Ethereum's implementation which returns the 0 address for errors.

The `castVoteBySig` function in the `GovernorAlpha` smart contract checks the return value is not address 0:

```
   function castVoteBySig(uint proposalId, bool support, uint8 v, bytes32 r, bytes32 s) public {
        bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(NAME)),
getChainId(), address(this)));
       bytes32 structHash = keccak256(abi.encode(BALLOT_TYPEHASH, proposalId, support));
       bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
       address signatory = ecrecover(digest, v, r, s);
       require(signatory != address(0), "GovernorAlpha::castVoteBySig: invalid signature");
       return _castVote(signatory, proposalId, support);
```

As a consequence, it is possible to cast a vote using an invalid signature. However, in the current implementation, the invalid voter would not have any voting power associated because he would lack the staking required.

The same scenario takes place in the `delegateBySig` function in Staking.sol:

```
       bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
       address signatory = ecrecover(digest, v, r, s);
       require(signatory != address(0), "Staking::delegateBySig: invalid signature");
```

Recommendation

Coinspect recommends checking for `ecrecover` RSK specific error return value in order to prevent future mistakes; for an example check RSKAddrValidator.sol.

SVN-015 Shadowed declarations

Total Risk	Impact	Location
Info	None	Staking.sol GovernorAlpha.sol
Fixed	Likelihood	
✘	None	

Description

The `Staking.sol` and `GovernorAlpha.sol` contracts emit warnings at compile time caused by shadowed declarations:

```
Staking.sol:393:13: Warning: This declaration shadows an existing declaration.
            uint96 currentBalance = currentBalance(account, i);

Staking.sol:228:5: The shadowed declaration is here:
    function currentBalance(address account, uint lockDate) internal view
returns(uint96) {

GovernorAlpha.sol:171:9: Warning: This declaration shadows an existing declaration.
        uint96 proposalThreshold = proposalThreshold();

GovernorAlpha.sol:153:5: The shadowed declaration is here:
    function proposalThreshold() public view returns (uint96) {

Staking.sol:393:13: Warning: This declaration shadows an existing declaration.
            uint96 currentBalance = currentBalance(account, i);

Staking.sol:228:5: The shadowed declaration is here:
    function currentBalance(address account, uint lockDate) internal view
returns(uint96) {

GovernorAlpha.sol:171:9: Warning: This declaration shadows an existing declaration.
        uint96 proposalThreshold = proposalThreshold();

GovernorAlpha.sol:153:5: The shadowed declaration is here:
    function proposalThreshold() public view returns (uint96) {
```

Coinspect auditors reviewed these warnings and concluded they do not represent an immediate risk.

Recommendation

Even though this issue does not represent a security risk right now, Coinspect recommends modifying the variable names to improve code readability and avoid the compile time warnings.

SVN-016 Incorrect revert error message strings in Checkpoints.sol

Total Risk	Impact	Location
Info	None	Checkpoints.sol
Fixed	Likelihood	
✘	None	

Description

The error strings used in several functions in the `Checkpoints` smart contract are incorrect, as the error is caused by an overflow in the stake amount and not in the stake date as the string indicates:

```
function _decreaseUserStake(address account, uint lockedTS, uint96 value) internal{
    uint32 nCheckpoints = numUserStakingCheckpoints[account][lockedTS];
    uint96 staked = userStakingCheckpoints[account][lockedTS][nCheckpoints - 1].stake;
    uint96 newStake = sub96(staked, value, "Staking::_decreaseUserStake: stakedUntil underflow");
    _writeUserCheckpoint(account, lockedTS, nCheckpoints, newStake);
```

The following error strings in `Checkpoints.sol` are incorrect:

```
uint96 newStake = add96(staked, value, "Staking::_increaseUserStake: stakedUntil overflow");
uint96 newStake = sub96(staked, value, "Staking::_decreaseUserStake: stakedUntil underflow");
uint96 newStake = add96(staked, value, "Staking::_increaseDelegateeStake: stakedUntil
overflow");
uint96 newStake = sub96(staked, value, "Staking::_decreaseDailyStake: stakedUntil underflow");
uint96 newStake = add96(staked, value, "Staking::_increaseDailyStake: stakedUntil overflow");
uint96 newStake = sub96(staked, value, "Staking::_decreaseDailyStake: stakedUntil underflow");
```

Recommendations

Modify the revert error strings to indicate a `newStake` overflow/underflow was the cause.

9. Disclaimer

The present security audit does not cover the endpoint systems and wallets that communicate with the contracts, nor the general operational security of the company whose contracts have been audited. This document should not be read as investment advice or an offering of tokens.