INX

Smart Contracts Audit

Oleksandr Zadorozhnyi, Roman Yarinskiy

August, 2023

Email: info@4irelabs.com

Website: 4irelabs.com

1. **INTRODUCTION.** INX requested 4irelabs a review of the contracts implementing their ERC1404 token. There are 6 contracts source code in scope (on commit eadded9):
   - *InxToken.sol*
   - *TransferRestrictionsNone.sol*
   - *TransferRestrictions.sol*
   - *RestrictionMessages.sol*
   - *IERC1404.sol*
   - *IERC1404Validators.sol*
2. **WARRANTY.** Audit is provided on an "as is" basis, without warranty of any kind, express or implied. The Auditor does not guarantee that the Code Review will identify all instances of security vulnerabilities or other related issues.
3. **EXECUTIVE SUMMARY.** 2 Lows and 4 Info issues have been found. Provided contracts realize ERC1404 mintable/burnable tokens with additional `TransferRestrictions` contracts that provide restriction functionality on token transfers. Contract admin could have the power to mint and burn any amount of tokens. Also, any user balance could be freely revoked or timelocked by the admin. Token transfers could be paused at any time. Only whitelisted users could receive and send tokens. There are no known compiler bugs, for the specified compiler versions, that might affect the logic of the contracts.

4. **CRITICAL BUGS AND VULNERABILITIES.** No Critical and Major issues have been found.

5. **LINE BY LINE REVIEW.**

5.1. InxToken.sol:

5.1.1. Line 73, 82. *Gas optimization*: modifiers `notRestricted` and `notRestrictedTransferFrom` read storage variable `_transferRestrictions` multiple times during execution.
Recommendation: Consider saving `_transferRestrictions` value in the local variables in the beginning of modifiers.

5.1.2. Line 140. *Info*: Consider moving zero address check from the `burn` function to the `setBurnAddress` function. This would prevent setting and emitting the wrong burn address and save some gas during each execution of the `burn` function.

5.1.3. Line 161. *Info*: Specs state that the `revoke` function should "increases the balance of the admin revoking the tokens", while function realization sends revoke tokens to the burning address.
Recommendation: Consider updating the revoke function or ignore this if the specs are outdated.

5.1.4. Line 225. *Low*: The `lock` function allows locking a number of tokens greater than the user balance while specs state that only "*all or part of a user's token balance until a specified time*".
Recommendation: Consider updating the `lock` function to check the current user balance before locking.

5.1.5. Line 251. *Gas optimization*: This line could be in block \`unchecked\` since \`amountToRelease\` would never be greater than \`lockedAmount\` during subtraction, so underflow couldn't happen.

5.1.6. Line 256. *Gas optimization*: This line could be in block \`unchecked\` since \`newLockedAmount\` would never be greater than \`lockedAmount\` during subtraction due to the logic of line 251, so underflow couldn't happen.

5.1.7. Line 288. *Low*: Since it is possible to lock more tokens than the current user balance - this could lead to underflow in this line and reverting with an unknown error.
Recommendation: Consider updating the \`checkTimelock\` function so it returns false in case \`lockupItem.amount\` > \`balance\`. This would result in a known error \`FAILURE_TIMELOCK_MESSAGE\` instead of an underflow revert.

5.1.8. Line 312. *Gas optimization:* Emitting the function parameter \`newRestrictionsAddress\` instead of the storage variable \`_transferRestrictions\` could save some gas.

5.1.9. *Gas optimization*: Consider using custom errors instead of reverting strings in \`require\` statements. Custom errors are supported since Solidity v0.8.4:https://soliditylang.org/blog/2021/04/21/custom-errors/

5.2. TransferRestrictionNone.sol:

5.2.1. *Line 6. Info:* Import of unused interface.

5.3. TransferRestriction.sol:

5.3.1. *Line 16. Gas optimization:* `validators` variable could be immutable since it never changes.

5.3.2. Line 64. *Gas optimization*: During the execution of the `detectTransferFromRestriction` function, the same `to` address would be checked to be whitelisted two times - on lines 36 and 64. *Recommendation*: Consider adding the `checkWhitelists` function with one address parameter and use it for checking the `sender` address alone.

5.3.3. Line 101. *Info*: `getSuccessCode` function returns `SUCCESS_CODE` with the type of uint256, while in other parts of the codebase `restrictionCode` values are uint8.