Streamr DATAv2 and Migration

Security Audit by

Version: 2

Authors:
Hristiyan Georgiev
Senior Blockchain Developer @ LimeChain

Date Created: **June 27, 2021**
Last Edit: **July 13, 2021**

Table of contents

Overview

The team behind Streamr followed good security practices, and it's shown in the code quality. The project is written and protected quite well from the common attacks. Pull mechanism has been chosen for the upgrade which is always the preferred way for doing transfers/payments, thus the risk of DoS(unexpected reverts, gasLimit, etc) attacks are eliminated.
The DATAv2 token is protected against ERC20 approval race, implementing standard ERC20 from OpenZeppelin. In addition the token is protected from Cross-chain replay attacks as well.
Code looks good with no critical / major issues found and is simple and easy to follow. Tests are covering completely the scenarios, and the coverage encompasses all lines which is commendable.

Methodology

The used methodology for the security audit consists of:

1. Understanding the purpose of the smart contracts;
2. Manual analysis on the smart contracts, covering the functional and security aspects;
3. Producing Audit Report containing all findings, potential problems and providing recommendations on how to resolve them.

Scope

GitHub Repository: https://github.com/streamr-dev/DATAv2
Git Commit: 8f97cf368b87e2c4d9a26672e7a5d33943d8fe93

File Name	SHA-1
contracts/DATAv2.sol	77a88dccfaf2bb9ae03e5106a5896e9079d8783f
contracts/DataTokenMigrator.sol	a78878a4ec45f7bb179afbb52d7a6feb99ce0ee6
test/DATAv2-test.js	0f1550652df12024ea84990ff1df3f8be850b5ae
test/DataTokenMigrator-test.js	8e9ec35364e854bcbf2d3f36d0caf3014bd1b9ca

Main points that were checked

- Front running attacks
- If malicious actor can harm the system
- Reentrancy issues
- Timestamp dependencies
- DoS attacks (gas)
- Griefings
- Overflows/underflows

Findings

DATAv2 Contract

- The token is protected against ERC20 approval race;
- The token is protected against Cross-chain replay attack;

#1 Resolved - Compiler Version (Low Impact, Low Probability)

The contract has an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Suggestion:
Consider that the compiler version is locked at the lowest version possible that the contract can be compiled at. For example, for version v0.8.0 the contract should contain the following line: pragma solidity 0.8.0;

#2 transferAndCall (Low Impact, Low Probability)

As this function allows contract to contract interactions, this is potentially vulnerable or open to a re-entrancy vulnerability, but we haven't found any exploit that could be used here causing harm to the contract.

Suggestion:
Since an access control is already implemented, it might be good IERC165 to be used on in order to make sure that the receiving contract supports interface of onTokenTransfer function before executing the actual transaction towards this contract (MockRecipient in this case). Since it's being casted to IERC677Receiver the assumption is that the contract has onTokenTransfer function but the function execution will continue nevertheless if it does not. By using the *supportInterface* pattern the transaction is going to be reverted if this is the case.

MockRecipientNotERC677Receiver has been added in order to test a recipient which does not implement *onTokenTransfer*. In order to better test this scenario, I would suggest adding a fallback to this contract. In result the first test in DATAv2-test.js (*transferAndCall triggers ERC677 callback*) will fail as the transaction on line #36 will be executed successfully. Not addressed. Not critical.

DataTokenMigrator Contract

#3 Resolved - Compiler Version (Low Impact, Low Probability)

The contract has an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Suggestion:
Consider that the compiler version is locked at the lowest version possible that the contract can be compiled at. For example, for version v0.8.0 the contract should contain the following line:
pragma solidity 0.8.0;

#4 Resolved - upgradeFrom function (Low Impact, Low Probability)

This is the only function intended for the transfers from the old token and is well protected

Suggestion:
This is a public function, but could be converted to external if it would not be called within the DataTokenMigrator instance itself. It will provide performance benefits and will potentially save on gas in future updates.