

Contract Deployment Verification



SpaceChain V2 - Contract Deployment Verification

Prepared for SpaceChain • September 2020

v210125

1. Executive Summary
2. Assessment
 - 2.1. Source Code Review
 - 2.2. Deployed Binary Code Verification
 - 2.3. Contract Initialization
3. Summary Of Findings
4. Findings
 - SPC-001 ERC20 Constructor Parameters Inverted
 - SPC-002 TokenUpgrader Address Hijack
5. Disclaimer

1. Executive Summary

In September 2020, SpaceChain engaged [Coinspect](#) to perform a security audit of the following SpaceChain Token V2 contract deployed to mainnet on [September 19th, 2020](#):

- SpaceChainV2 (**SPC**)
<https://etherscan.io/address/0x86ed939b500e121c0c5f493f399084db596dad20>

The objective of the audit was to evaluate the security of the smart contract source code, deployment, and user tokens migration procedures. During the assessment, Coinspect identified the following issues:

High Risk	Medium Risk	Low Risk
0	1	0

User tokens could be lost during the migration process if special care is not taken, Coinspect recommends this fact is properly documented in the instructions provided to token owners.

The following report details the tasks performed during this engagement and its findings.

2. Assessment

2.1. Source Code Review

The goal of the new SpaceChain token V2 is to replace the existing SpaceChain ERC20 token on Ethereum ([Contract Address 0x8069080a922834460c3a092fb2c1510224dc066b](#)) in order to better support the ERC20 standard interface with the purpose of being listed by many decentralized financial products. These are the features added to the new token version:

- Approve
- Increase, decrease allowance
- TransferFrom
- Bool return values

Coinspect audited the new token's source code and evaluated the proposed upgrade mechanism to determine if it is safe for the users migrating their tokens to the new version.

The assessment covered the master branch of the repository at <https://github.com/spacechain/token-v2> up to (and including) commit `bd8f266c1d99cce4e222a0bae4076df4ede80746` of **September 21st, 2020**:

```
commit bd8f266c1d99cce4e222a0bae4076df4ede80746
Author: Jeff Garzik <jeff@bloq.com>
Date:   Mon Sep 21 19:24:30 2020 -0400
```

```
SpaceChain.sol: fix ERC20 constructor call
```

The scope of the audit was limited to the following Solidity source files, shown here with their sha256sum hash:

```
3f581f265049d9718fa0ebb967443018481eee694c4c208ffb92209ebc5f17d2 Migrations.sol
78c265bfc5ebef8e559366663225a22aca9b3b97bfa5fc4a92bd5b48db8c2a62 SpaceChain.sol
aa9ed1f1adc849598b257e8786153d8db31b1b205ab3e7d0057aad8b61294314 TokenUpgrader.sol
```

A whitebox security audit was conducted on these smart contracts. The following threat scenarios were evaluated during this audit:

- Are the user tokens safe during the migration process?
- Does the new SpaceChain Token version alter the owner's privileges in comparison to the previous version?
- Is there any way to inflate the token supply with the new version?
- Are the migration instructions provided to the end-user safe?

The token migration process is non-custodial and voluntary. Users choose when and how many tokens they want to upgrade. Users are encouraged to upgrade first small amounts of SPC in order to make sure they understand the process.

It is worth noting that SPC v1 tokens are upgraded to SPC v2 tokens on a 1:1 basis, the token supply remains capped at 1 billion.

The token migration procedure reviewed during this engagement is described in <https://github.com/spacechain/token-v2/blob/master/ETHERSCAN-MIGRATION.md>. SpaceChain also intends to provide users with a website to facilitate the procedure, this website was not part of this audit.

According to the above mentioned documentation the following opt-in upgrade process should be performed by users that wish to upgrade their SPC v1 tokens to the new version:

1. Call the `createUpgrader()` function in the new token contract to create a user-owned vault contract. (*Auditor's note: the contract is actually not user-owned*)
2. User sends SPC v1 tokens to the address of their vault contract.
3. User calls `migrateV1tokens()` function in the new token contract, which mints (credits) SPC v2 tokens, and burns (debits) SPC v1 tokens from the vault contract.

Coinspect found that if tokens could be lost during a chain reorganization if inexperienced users fail to wait enough confirmations before sending their tokens to the new vault. It is suggested to document this potential problem to prevent this from happening. This finding is fully detailed in [User tokens lost during migrations because of blockchain reorganizations](#).

With regards to the test provided, it was limited to a simple use scenario. Coinspect suggests in the future more tests are developed before deployment. For example, a test should have been included to verify a new vault was not accessible from a third account. Coinspect was not able to verify the provided test passed as it requires an account with SpaceChainV1 tokens balance in Mainnet. Also, the test provided contained strings which did not correspond to this project.

2.2. Deployed Binary Code Verification

As a part of the additional tasks performed to ensure the SpaceChainV2 token contract was deployed safely, Coinspect verified the deployed smart contract bytecode, in order to guarantee no backdoors or unintended vulnerabilities were introduced.

Coinspect downloaded the contract creation bytecode from the blockchain and compared it with the output of the `solc` solidity compiler version `0.6.12+commit.27d51765`, using 200 optimization runs. The metadata was removed from the bytecode before comparing the sha256 hashes of the locally compiled code and the code deployed:

```
f89f8affbbefb69ee68831bcea870ac3035fa53131a8e6b7abfa62cea6bc6538 SpaceChain.solc.nometadata
f89f8affbbefb69ee68831bcea870ac3035fa53131a8e6b7abfa62cea6bc6538 SpaceChain.creationBytecode.nometadata
```

Initially, Coinspect found the locally compiled contract did not match the version deployed in the blockchain. This was caused because the version in the repository had the parameters to the ERC20 constructor inverted. This is detailed in [ERC20 constructor parameters inverted](#). After fixing this we were able to verify the deployed contract matched the source code in the repository.

2.3. Contract Initialization

The new SpaceChainV2 token contract address is:

[0x86ed939b500e121c0c5f493f399084db596dad20](#)

This contract was deployed by:

[0x60f7e71e244b83e5c679cc16727ae1a09243a8f2](#)

Contract creation transaction:


[0x5f6b183b8384d8b4b16565bdc47c5b8f4fcb8b73e0a51074e243c1c3e88a4eb8](#)

The smart contract behavior is not mandated by any parameters supplied during deployment, nor does require any additional initialization steps after being created.

3. Summary Of Findings

ID	Description	Risk	Fixed
SPC-001	ERC20 Constructor Parameters Inverted	Info	✓
SPC-002	TokenUpgrader Address Hijack	Medium	✓

4. Findings

SPC-001 ERC20 Constructor Parameters Inverted		
Total Risk Info	Impact Low	Location SpaceChain.sol
Fixed 	Likelihood Low	addressed in commit bd8f266c1d99cce4e222a0bae4076df4ede80746

Description

The SpaceChain token source code in the repository calls the ERC20 constructor function with its parameter in inverted order: the constructor expects the token name first, and then the corresponding symbol.

```
contract SpaceChain is ERC20Burnable {  
    mapping(address => TokenUpgrader) public tokenUpgrader;  
  
    constructor() public ERC20("SPC","SpaceChainV2") {}  
}
```


Coinspect noticed the smart contract deployed to Ethereum mainnet has the parameters set in the proper order, however these changes have not been committed to the public repository yet.

Recommendations

Commit the changes performed before deployment to the repository. The contract should read:

```
constructor() public ERC20("SpaceChainV2","SPC") {}
```


SPC-002 TokenUpgrader Address Hijack

Total Risk Medium	Impact High	Location SpaceChain.sol
Fixed 	Likelihood Medium	addressed in commit b5d4f27211790acc9fa3e8ca13d5644f2dee3abc ,

Description

The token migration procedure involves the user transferring his tokens to the Upgrader contract. This contract is created when the user calls the `createUpgrader` function in the new token contract:

```
function createUpgrader() public {  
    require(  
        address(tokenUpgrader[_msgSender()]) == address(0x0),  
        "Already created upgrader contract"  
    );  
    tokenUpgrader[_msgSender()] = new TokenUpgrader(_msgSender());  
}
```

The user then must query the SpaceChain contract to obtain his TokenUpgrader address, and transfer his tokens to this address.

If a **chain reorganization** takes place between the moment the user obtains the contract address and the moment tokens are transferred, these tokens could end up being transferred to **a TokenUpgrader contract with the same address but a different tokenHolder**, which would then own these tokens.

Recommendations

Consider adding a visible warning to users letting them know tokens are at risk if not enough block confirmations take place after the creation of their new Upgrader contract and before the transfer of funds. Also, make sure the website that will be made available to help users migrate their tokens is implemented in a way that enforces a minimum number of block confirmations.

Status

This issue was addressed in commit [b5d4f27211790acc9fa3e8ca13d5644f2dee3abc](#), which adds a warning for users to wait enough confirmations in the migration documentation.

5. Disclaimer

The information presented in this document is provided as is and without warranty. Vulnerability assessments are a “point in time” analysis and as such it is possible that something in the environment could have changed since the tests reflected in this report were run. This report should not be considered a perfect representation of the risks threatening the analysed system, networks and applications.