Smart Contract Source Code Audit

Prepared for SWARM • October 2019

v191023

1. Table Of Contents

2. Executive Summary

In October 2019, Swarm engaged Coinspect to perform a source code review of the Swarm compliant token contracts. This is a follow-up of an audit of an older version of the smart contracts performed by Coinspect in June 2019. The objective of the audit was to evaluate the security of the smart contracts.

During the assessment, Coinspect found no critical security issues, and only two low risk issues. SWM-001 calls attention to a missing check for the return value of a transfer function, and SWM-002 involves a private key that has been committed to the public Git repository.

The code was found to be well documented and very readable, following good coding practices except for small details like SWM-003 and some compilation warnings (they are inconsequential to security but it is recommended to eliminate them).

The repository also includes 60 tests (40 for `SRC20`, 17 for `SRC20Factory`, and 3 for `GetRateMinter`). All tests pass without problems, but they don't have full coverage. It is recommended to implement more tests and make sure they have good coverage.

It is worth mentioning that the contracts are not fully autonomous and rely on transactions by accounts controlled by Swarm.

3. Introduction

The Swarm compliant token contracts include the SRC20 token and additional contracts that implement a factory that Swarm uses to create new SRC20 tokens, a registry of SRC20 tokens and managers that handle SRC20 burn/mint in relation to SWM token staking, an oracle for querying the price of SWM in USD, an assets registry that hold properties of off-chain assets being tokenized, and auxiliary contracts for features of the SRC20 tokens such as transfer restrictions, freezability, roles (authority, delegate), and transfer rules (whitelisting and gray listing).

Swarm specified that the audit was to be conducted on the latest smart contracts in the branch `develop` of the Git repository at `https://github.com/swarmfund/swarm-compliant-contract.git` as of commit `d5484b02f20505b6732e9f5e65d766ed921b733b` of October 8.

The issues identified during the assessment were fixed in commit `91eabdc1b06a5354687b6bba55c8d7ba06adeb51` of October 16.

The scope of the audit was limited to the following Solidity source files, shown here with their sha256sum hash as of commit `91eabdc1b06a5354687b6bba55c8d7ba06adeb51`:

```
1c4e30fd3aa765cb0ee259a29dead71c1c99888dcc7157c25df3405802cf5b09  Migrations.sol
69f0c3dbbc363a365abf5a2af66b06b059c03087aeba35d5fff09d8429a5a9b0  factories/GetRateMinter.sol
9fb20c199e87bbc28034ba0f433a66899d8ddb0b23f1f6e00f8241757f6ea3df  factories/Manager.sol
0cab34a6af271119b9ad9a0bf70e77b89d428f345de1cd5b76bd75f53abb1cb0  factories/SetRateMinter.sol
f4e9e419d40598d13fdd55b4a773e5e565aa70ace9bee316948647743c57fb92  factories/SRC20Factory.sol
f99089972708f282bf10c9b8e1e89edddd51bb60fe989835ccfd5ccb4cc2d44d  factories/SRC20Registry.sol
e7d853bd0991a81ac8bb5bc870e0a9eba9b35cbb6da29eca31f689fa89a568d8  factories/SWMPriceOracle.sol
9567a79298ee92906d01816307bba92f4335e1441aac030b932572fbadbe9ee2  interfaces/IAssetRegistry.sol
c0e320e2c3381aea2a96d7ce4c732881c0488dffa16405b128c52467a9b6dfe8  interfaces/IFeatured.sol
e4aaa477f1de8d5cd6e5f8948c55f285519da582b6bf259cda6f2a1b7454c069  interfaces/IFreezable.sol
9569bd38f9fd699f15605b2756249c45ed3324a23edf1a7f70c60408ddf22ead  interfaces/IManager.sol
247c58310c3f5a0e7f24013ac1f270dbfb8863704dd7d3e52c8ad480bbab0224  interfaces/INetAssetValueUSD.sol
049ab43727feaf308957c7b3b5cba290f33e1c54ace5b53e308b2e5c8aa01cf7  interfaces/IPausable.sol
3f4737cc649ad8dd8e47f176d04c2e501ae1e4edb8eebb7ad9a112495aa9af99  interfaces/IPriceUSD.sol
cb443b4326ff8b50d50f6135994944b9df426fe8e802b2ad4b62b5a940710dd2  interfaces/ISRC20Managed.sol
0f6cbc39e794f86b05fc5d46e79caedefb0a94c45c44950f82b1a4e0ce222119  interfaces/ISRC20Registry.sol
a6b65aa97832e35ca4f4edc15e7fa4bd58d5ba8fe48d6e24b709dbf3b1d2cd34  interfaces/ISRC20Roles.sol
685708b2afe500d1af38b8c04b2cc7815dbf0a776ab64ffcb19b5c35f682ce34  interfaces/ISRC20.sol
cfe0d003a7986e8d5cd69210b186d3389d2ec755a59967925e905f1940badeb8  interfaces/ITransferRestrictions.sol
a7368564f369c1a3757ef73cff7184043ab188ef2c992d7a20d1d664b4efbb08  interfaces/ITransferRules.sol
fb665c577224da90ef477e594af7d4d2a7368bd77425c9a62e439738d095bbe6  roles/AuthorityRole.sol
d0128deeff289567e9563a8dfadf5bdb81df15b5d9ede88e29905b57b5d80c09  roles/DelegateRole.sol
248a9e4ec7d3ec4d79c8667f25f6bbeb6336c0dbf72fd8c8e3ccd653d3f79add  roles/Managed.sol
d1f9fd2a261d4844a1476a61389306d761308f8d1aae3b7d3ce95cecd79f4664  roles/SRC20Roles.sol
81e9dcaaa91891facf4ee1194343afaecb4c18b8d89b8e74a935998f782e17ee  rules/ManualApproval.sol
1b8f01c62a10376c9d43b7aa4180c34fe8d317d1df2d2cb2bad822e6f821286d  rules/TransferRules.sol
66ec772aa52de7d70a1558a1a6716c39d66101f590e54566c044e935e7334097  rules/Whitelisted.sol
27c8a73563c22d1ad9c923d04c9c9c743d649ba7b96a155f0f7a0a0022fd8f9a  token/AssetRegistry.sol
b1dedb6016dbc9e5d6645199b1df6c8da648f30ff6c8393f3a511f669d95ce8a  token/SRC20Detailed.sol
ba7436cee3292734a0d9d8902d071bbf109a2f5440dd84309f10f64cb62c185f  token/SRC20.sol
6633e21ae3ade222e16f19c18022762ac9f666b63529e061e96feda078e6c8de  token/features/Featured.sol
4a7d3fd22eef5f429bba84ee2df25615d8d76f61645bd2f4a0528ad1ef21c816  token/features/Freezable.sol
14d54e9bcd0bcce493e6ee2576c4cd21c7fb14fb9e25e75915b20b5182ec0957  token/features/Pausable.sol
```

4. Summary Of Findings

ID	Description	Risk	Fixed
SWM-001	Ignored return value of executeTransfer could lead to locked funds	Low	✔
SWM-002	Private key found in public Git repository	Low	✔
SWM-003	Missing error message in calls to require()	Zero	✔

5. Findings

SWM-001	Ignored return value of executeTransfer could lead to locked funds	
Total Risk **Low**	Impact High	Location ManualApproval.sol
Fixed ✔	Likelihood Low	

Description

The contract `ManualApproval` implements a form of gray listing. A transfer from a gray listed account 'from' to another account 'to' is not performed immediately, and instead the funds are first transferred to the contract while the request for transfer awaits the explicit approval of the owner of the contract:

```
function _transferRequest(address from, address to, uint256 value) internal
returns (bool) {
        require(_src20.executeTransfer(from, address(this), value), "SRC20
transfer failed");

        _transferReq[_reqNumber] = TransferReq(from, to, value);

        emit TransferRequest(_reqNumber, from, to, value);
        _reqNumber = _reqNumber + 1;

        return true;
}
```

The owner can then call the function `transferApproval(uint256)` to approve and perform the transfer request by transferring the funds from the contract to 'to':

```
function transferApproval(uint256 reqNumber) external onlyOwner returns
(bool) {
        TransferReq memory req = _transferReq[reqNumber];

        require(_src20.executeTransfer(address(this), req.to, req.value),
"SRC20 transfer failed");

        delete _transferReq[reqNumber];
        emit TransferApproval(reqNumber, req.from, req.to, req.value);
        return true;
}
```

And before the owner approves the transfer and while the funds are still being held in the contract the ‘from’ account can call the function `cancelTransferRequest(uint256)` to cancel the transfer and get the funds back:

```
function cancelTransferRequest(uint256 reqNumber) external returns (bool) {
      TransferReq memory req = _transferReq[reqNumber];
      require(req.from == msg.sender, "Not owner of the transfer request");

      _src20.executeTransfer(address(this), req.from, req.value);

      delete _transferReq[reqNumber];
      emit TransferRequestCanceled(reqNumber, req.from, req.to, req.value);

      return true;
}
```

Notice that the function `cancelTransferRequest(uint256)` ignores the return value of the function `executeTransfer(address,address,uint256)`, while the functions `_transferRequest(address,address,uint256)` and `transferApproval(uint256)` both require the return value of `executeTransfer(address,address,uint256)` to be true and otherwise revert.

Currently this inconsistency is not a threat, because the implementation of the function `executeTransfer(address,address,uint256)` in the `SRC20` contract either succeeds and returns true or reverts. But, if for some reason the transfer failed returning false, the failure would be unnoticed by the function `cancelTransferRequest(uint256)` and the funds would be locked in the contract.

Recommendation

It is advisable to resolve this inconsistency and prevent future problems simply by making the function `cancelTransferRequest(uint256)` also require the return value to be true.

SWM-002 Private key found in public Git repository

Total Risk Low Fixed ✔	Impact Low Likelihood High	Location https://github.com/swarmfund/swarm-compliant-contract/blob/develop/.private_key

Description

A private key used for deploying contracts through the Infura system was found in the public Git repository in file `.private_key`. The file is present in branches `develop` and `feature/contract-mocks`.

The file `.private_key` containing the private key, as well as the file `.secret` containing the mnemonic to generate the key, are not intended to be committed to Git and as such they were both already listed in `.gitignore`.

When made aware of the issue Swarm communicated to Coinspect that the key has not been used for anything critical, and was already considered unsafe because it had been sent between developers in unencrypted emails.

Recommendation

Remove the private key from the Git repository and make sure it is not used again for anything critical, since the key is already exposed. Also, keep in mind that storing secrets in the file system is in general not a good practice and could lead to compromise. Consider the use of HSM (hardware security module).

SWM-003 Missing error message in calls to require()

Total Risk	Impact	Location
Zero	Zero	TransferRules.sol, Managed.sol, SRC20.sol, Manager.sol
Fixed ✔	Likelihood Low	

Description

Some calls to `require()` are missing an an error string in case the required condition is not met. The following are all the instances of the issue:

```
TransferRules.sol:17:
        require(msg.sender == address(_src20));
Managed.sol:55:
        require(newManager != address(0));
SRC20.sol:339:
        require(_features.checkTransfer(msg.sender, to));
SRC20.sol:351:
        require(_features.checkTransfer(from, to));
SRC20.sol:428:
        require(_features.checkTransfer(from, to));
SRC20.sol:441:
        require(to != address(0));
Manager.sol:85:
        require(_swmERC20.transferFrom(swmAccount, address(this), swmValue));
Manager.sol:86:
        require(ISRC20Managed(src20).mint(_registry[src20].owner, src20Value));
Manager.sol:116:
        require(_swmERC20.transferFrom(swmAccount, address(this), swmValue));
Manager.sol:117:
        require(ISRC20Managed(src20).mint(_registry[src20].owner,         _calcTokens(src20,
swmValue)));
Manager.sol:148:
        require(_swmERC20.transfer(swmAccount, swmValue));
Manager.sol:149:
        require(ISRC20Managed(src20).burn(_registry[src20].owner,         _calcTokens(src20,
swmValue)));
Manager.sol:169:
        require(ISRC20Roles(_registry[src20].roles).renounceManagement());
```

Since Solidity version 0.4.22 `require()` and `assert()` support a reason string as second parameter. It is a good practice to always include a reason string in any `require()` or `assert()`, so if a requirement or assertion is not met an informative error string would be found in the transaction receipt.

Recommendation

Add an error message string to the `require()` calls that don't have one.

6. Disclaimer

The present security audit is limited to smart contract code. It does not cover the technologies and designs related to these smart contracts, nor the frameworks 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.