CODE REVIEW ASSIGNMENT

Stephen Laz-Eke

07-10-2022

I was given MakerDAO's flash mint contract (https://github.com/makerdao/dss-flash/blob/master/src/flash.sol) to review.

The flash mint process allows users to buy tokens but pay for them at the end of the transaction.

Now into the code:

183 lines (150 sloc) | 6.12 KB

```
1   // SPDX-License-Identifier: AGPL-3.0-or-later
2   // Copyright (C) 2021 Dai Foundation
3   //
4   // This program is free software: you can redistribute it and/or modify
5   // it under the terms of the GNU Affero General Public License as published by
6   // the Free Software Foundation, either version 3 of the License, or
7   // (at your option) any later version.
8   //
9   // This program is distributed in the hope that it will be useful,
10  // but WITHOUT ANY WARRANTY; without even the implied warranty of
11  // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  // GNU Affero General Public License for more details.
13  //
14  // You should have received a copy of the GNU Affero General Public License
15  // along with this program.  If not, see <https://www.gnu.org/licenses/>.
16
17  pragma solidity 0.6.12;
18
19  import "./interface/IERC3156FlashLender.sol";
20  import "./interface/IERC3156FlashBorrower.sol";
21  import "./interface/IVatDaiFlashLender.sol";
22
```

It is a 183-line codebase of 6.12KB size.

It uses the AGPL-3.0 (Affero General Public License) which mandates making any modification of this source code that is used by the public open-source.

It uses solidity compiler version 0.6.12 as it was the current version in 2020, when it was first written.

It imports three interfaces, two of which are of ERC3156 which provides standard interfaces and processes for single-asset flash loans.

The interfaces are reviewed next.

1. The IERC3156FlashLender.sol

56 lines (50 sloc) | 1.93 KB

```sol
// SPDX-License-Identifier: AGPL-3.0-or-later
// Copyright (C) 2021 Dai Foundation
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Affero General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Affero General Public License for more details.
//
// You should have received a copy of the GNU Affero General Public License
// along with this program.  If not, see <https://www.gnu.org/licenses/>.

pragma solidity >=0.6.12;

import "./IERC3156FlashBorrower.sol";

interface IERC3156FlashLender {

    /**
     * @dev The amount of currency available to be lent.
     * @param token The loan currency.
     * @return The amount of `token` that can be borrowed.
     */
    function maxFlashLoan(
        address token
    ) external view returns (uint256);

    /**
     * @dev The fee to be charged for a given loan.
     * @param token The loan currency.
     * @param amount The amount of tokens lent.
     * @return The amount of `token` to be charged for the loan, on top of the returned principal.
     */
    function flashFee(
        address token,
        uint256 amount
    ) external view returns (uint256);

    /**
     * @dev Initiate a flash loan.
     * @param receiver The receiver of the tokens in the loan, and the receiver of the callback.
     * @param token The loan currency.
     * @param amount The amount of tokens lent.
     * @param data Arbitrary data structure, intended to contain user-defined parameters.
     */
    function flashLoan(
        IERC3156FlashBorrower receiver,
        address token,
        uint256 amount,
        bytes calldata data
    ) external returns (bool);
}
```

All its functions are implemented in the contract.

2. The IERC3156FlashBorrower.sol

37 lines (34 sloc) | 1.38 KB

```
1   // SPDX-License-Identifier: AGPL-3.0-or-later
2   // Copyright (C) 2021 Dai Foundation
3   //
4   // This program is free software: you can redistribute it and/or modify
5   // it under the terms of the GNU Affero General Public License as published by
6   // the Free Software Foundation, either version 3 of the License, or
7   // (at your option) any later version.
8   //
9   // This program is distributed in the hope that it will be useful,
10  // but WITHOUT ANY WARRANTY; without even the implied warranty of
11  // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  // GNU Affero General Public License for more details.
13  //
14  // You should have received a copy of the GNU Affero General Public License
15  // along with this program.  If not, see <https://www.gnu.org/licenses/>.
16
17  pragma solidity >=0.6.12;
18
19  interface IERC3156FlashBorrower {
20
21      /**
```

```
19  interface IERC3156FlashBorrower {
20
21      /**
22       * @dev Receive a flash loan.
23       * @param initiator The initiator of the loan.
24       * @param token The loan currency.
25       * @param amount The amount of tokens lent.
26       * @param fee The additional amount of tokens to repay.
27       * @param data Arbitrary data structure, intended to contain user-defined parameters.
28       * @return The keccak256 hash of "ERC3156FlashBorrower.onFlashLoan"
29       */
30      function onFlashLoan(
31          address initiator,
32          address token,
33          uint256 amount,
34          uint256 fee,
35          bytes calldata data
36      ) external returns (bytes32);
37  }
```

It contains only one function that is implemented in the tests, but called in the contract.

3. The IVatDaiFlashLender.sol

34 lines (30 sloc) | 1.27 KB

```solidity
// SPDX-License-Identifier: AGPL-3.0-or-later
// Copyright (C) 2021 Dai Foundation
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Affero General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Affero General Public License for more details.
//
// You should have received a copy of the GNU Affero General Public License
// along with this program.  If not, see <https://www.gnu.org/licenses/>.

pragma solidity >=0.6.12;

import "./IVatDaiFlashBorrower.sol";

interface IVatDaiFlashLender {

    /**
     * @dev Initiate a flash loan.
     * @param receiver The receiver of the tokens in the loan, and the receiver of the callback.
     * @param amount The amount of tokens lent. [rad]
     * @param data Arbitrary data structure, intended to contain user-defined parameters.
     */
    function vatDaiFlashLoan(
        IVatDaiFlashBorrower receiver,
        uint256 amount,
        bytes calldata data
    ) external returns (bool);
}
```

A function for VatDai flash loans which is implemented in the contract.

Notice that there is no token parameter because it is vatDai by default.

Back to the main code now, let’s look at the other lines.

Three more interfaces are defined, as a model for the three types of smart contracts used:

```
23  interface DaiLike {
24      function balanceOf(address) external returns (uint256);
25      function transferFrom(address, address, uint256) external returns (bool);
26      function approve(address, uint256) external returns (bool);
27  }
28
29  interface DaiJoinLike {
30      function dai() external view returns (address);
31      function vat() external view returns (address);
32      function join(address, uint256) external;
33      function exit(address, uint256) external;
34  }
35
36  interface VatLike {
37      function hope(address) external;
38      function dai(address) external view returns (uint256);
39      function live() external view returns (uint256);
40      function move(address, address, uint256) external;
41      function heal(uint256) external;
42      function suck(address, address, uint256) external;
43  }
```

They will be explained subsequently.

The contract then inherits two of the interfaces that were imported and implements their functions:

```
44
45  contract DssFlash is IERC3156FlashLender, IVatDaiFlashLender {
46
```

It then defines some datatypes for authentication:

```
47      // --- Auth ---
48      function rely(address usr) external auth { wards[usr] = 1; emit Rely(usr); }
49      function deny(address usr) external auth { wards[usr] = 0; emit Deny(usr); }
50      mapping (address => uint256) public wards;
51      modifier auth {
52          require(wards[msg.sender] == 1, "DssFlash/not-authorized");
53          _;
54      }
```

It represents Boolean ‘true’ as ‘1’ and ‘false’ as ‘0’.
The rely function authorizes the contract for an address and the deny function deauthorizes.

Some more datatypes for the operations:

```
55
56      // --- Data ---
57      VatLike     public immutable vat;
58      DaiJoinLike public immutable daiJoin;
59      DaiLike     public immutable dai;
60
61      uint256     public  max;     // Maximum borrowable Dai  [wad]
62      uint256     private locked;  // Reentrancy guard
63
64      bytes32 public constant CALLBACK_SUCCESS = keccak256("ERC3156FlashBorrower.onFlashLoan");
65      bytes32 public constant CALLBACK_SUCCESS_VAT_DAI = keccak256("VatDaiFlashBorrower.onVatDaiFlashLoan");
66
```

Instances of the defined interfaces are created.

The max variable is in WAD unit is a fixed point decimal with 18 decimals (usually for basic quantities, e.g. balances).

Constant encrypted data are defined and used for harmonization/standardization and security.

Next, the events and re-entrancy guard modifier are defined:

```
67      // --- Events ---
68      event Rely(address indexed usr);
69      event Deny(address indexed usr);
70      event File(bytes32 indexed what, uint256 data);
71      event FlashLoan(address indexed receiver, address token, uint256 amount, uint256 fee);
72      event VatDaiFlashLoan(address indexed receiver, uint256 amount, uint256 fee);
73
74      modifier lock {
75          require(locked == 0, "DssFlash/reentrancy-guard");
76          locked = 1;
77          _;
78          locked = 0;
79      }
80
```

The constructor is defined next, which takes in the address of the DaiJoin contract that allows users to withdraw their Dai from the system into a standard ERC20 token:

```
81      // --- Init ---
82      constructor(address daiJoin_) public {
83          wards[msg.sender] = 1;
84          emit Rely(msg.sender);
85
86          VatLike vat_ = vat = VatLike(DaiJoinLike(daiJoin_).vat());
87          daiJoin = DaiJoinLike(daiJoin_);
88          DaiLike dai_ = dai = DaiLike(DaiJoinLike(daiJoin_).dai());
89
90          vat_.hope(daiJoin_);
91          dai_.approve(daiJoin_, type(uint256).max);
92      }
93
94      // --- Math ---
95      uint256 constant RAY = 10 ** 27;
```

The hope and approve functions are defined in the test contract.

Next, constant units and the multiplication function are defined:

RAY and RAD are DAI units of 27 and 45 decimals respectively.

Next, the file function used in the tests for implementing an upper-limit:

```
101     // --- Administration ---
102     function file(bytes32 what, uint256 data) external auth {
103         if (what == "max") {
104             // Add an upper limit of 10^27 DAI to avoid breaking technical assumptions of DAI << 2^256 - 1
105             require((max = data) <= RAD, "DssFlash/ceiling-too-high");
106         }
107         else revert("DssFlash/file-unrecognized-param");
108         emit File(what, data);
109     }
110
```

Next, the ERC 3156 specifications in the inherited interfaces are implemented:

```
110
111     // --- ERC 3156 Spec ---
112     function maxFlashLoan(
113         address token
114     ) external override view returns (uint256) {
115         if (token == address(dai) && locked == 0) {
116             return max;
117         } else {
118             return 0;
119         }
120     }
121
122     function flashFee(
123         address token,
124         uint256 amount
125     ) external override view returns (uint256) {
126         amount;
127         require(token == address(dai), "DssFlash/token-unsupported");
128
129         return 0;
130     }
131
```

Then the main function, the flash loan is defined:

```
132     function flashLoan(
133         IERC3156FlashBorrower receiver,
134         address token,
135         uint256 amount,
136         bytes calldata data
137     ) external override lock returns (bool) {
138         require(token == address(dai), "DssFlash/token-unsupported");
139         require(amount <= max, "DssFlash/ceiling-exceeded");
140         require(vat.live() == 1, "DssFlash/vat-not-live");
141
142         uint256 amt = _mul(amount, RAY);
143
144         vat.suck(address(this), address(this), amt);
145         daiJoin.exit(address(receiver), amount);
146
147         emit FlashLoan(address(receiver), token, amount, 0);
148
149         require(
150             receiver.onFlashLoan(msg.sender, token, amount, 0, data) == CALLBACK_SUCCESS,
151             "DssFlash/callback-failed"
152         );
```

```
152         );
153
154         dai.transferFrom(address(receiver), address(this), amount);
155         daiJoin.join(address(this), amount);
156         vat.heal(amt);
157
158         return true;
159     }
```

The ‘flashLoan’ function is what is called to take a flash loan. It then calls the ‘onflashLoan’ function defined in the tests which does the minting and returns the completion status.

Next, the VatDai flash loan function is defined.
Vat is the core Vault engine of the Dai Stablecoin System (DSS). It stores Vaults and tracks all the associated Dai and Collateral balances. It also defines the rules by which Vaults and balances can be manipulated. The rules defined in the `Vat` are immutable, so in some sense, the rules in the `Vat` can be viewed as the constitution of `dss`. (makerdao docs)

This function does not take in a token parameter, as it is VatDai by default.

```
162     function vatDaiFlashLoan(
163         IVatDaiFlashBorrower receiver,          // address of conformant IVatDaiFlashBorrower
164         uint256 amount,                         // amount to flash loan [rad]
165         bytes calldata data                     // arbitrary data to pass to the receiver
166     ) external override lock returns (bool) {
167         require(amount <= _mul(max, RAY), "DssFlash/ceiling-exceeded");
168         require(vat.live() == 1, "DssFlash/vat-not-live");
169
170         vat.suck(address(this), address(receiver), amount);
171
172         emit VatDaiFlashLoan(address(receiver), amount, 0);
173
174         require(
175             receiver.onVatDaiFlashLoan(msg.sender, amount, 0, data) == CALLBACK_SUCCESS_VAT_DAI,
176             "DssFlash/callback-failed"
177         );
178
179         vat.heal(amount);
180
181         return true;
182     }
183 }
```

That is the end of the contract.

Some of the functions it calls are defined in the test file used to deploy the contract.

OBSERVATIONS AND RECOMMENDATIONS

The code was not updated to use some breaking changes from 0.7.0 or 0.8.0, a classic case of ‘if it isn’t broke, don’t fix it’. The code works fine and there is also no need for any lowlevel interaction.

Therefore, I think the code is okay and I have no recommendations for it.

Thank you.