Flash Mint Module

Enables any user to execute a flash mint of Dai. * Contract Name: * flash.sol * Type/Category: * DSS *Contract Source* * Etherscan *

1. Introduction (Summary)

TheFlash module allows anyone to mint Dai up to a limit set by Maker Governance with the one condition that they pay it all back in the same transaction with a fee. This allows anyone to exploit arbitrage opportunities in the DeFi space without having to commit upfront capital.Flash provides many benefits to the Dai ecosystem including, but not limited to:

- Improved market efficiencies for Dai.
- Democratization of arbitrage - anyone can participate.
- Exploits requiring a large amount of capital will be found quicker which makes the DeFi space safer overall.
- Fees provide an income source for the protocol.
-

1. Contract Details

Glossary (Flash)

- Debt Ceiling
- : The maximum amount of Dai any single transaction can borrow. Encoded asline
- inrad
- units.
- Minting Fees
- : How much additional Dai must be returned to theFlash
- module at the end of the transaction. This fee is transferred into thevow
- at the end of a successfulmint
- . Encoded astoll
- inwad
- units.
-

1. Key Mechanisms & Concepts

Usage

Since the Flash module conforms to the ERC3156 spec, you can just use the reference borrower implementation from the spec:

```

Copy pragma solidity ^0.8.0;

import "./interfaces/IERC20.sol"; import "./interfaces/IERC3156FlashBorrower.sol"; import "./interfaces/IERC3156FlashLender.sol";

contract FlashBorrower is IERC3156FlashBorrower { enum Action {NORMAL, OTHER}

IERC3156FlashLender lender;

constructor ( IERC3156FlashLender lender_ ) public { lender = lender_; }

/// @dev ERC-3156 Flash loan callback function onFlashLoan( address initiator, address token, uint256 amount, uint256 fee, bytes calldata data ) external override returns (bytes32) { require( msg.sender == address(lender), "FlashBorrower: Untrusted lender" ); require( initiator == address(this), "FlashBorrower: Untrusted loan initiator" ); (Action action) = abi.decode(data, (Action)); if (action == Action.NORMAL) { require(IERC20(token).balanceOf(address(this)) >= amount); // make a profitable trade here IERC20(token).transfer(initiator, amount + fee); } else if (action == Action.OTHER) { // do another } return keccak256("ERC3156FlashBorrower.onFlashLoan"); }

/// @dev Initiate a flash loan function flashBorrow( address token, uint256 amount ) public { bytes memory data = abi.encode(Action.NORMAL); uint256 _allowance = IERC20(token).allowance(address(this), address(lender)); uint256 _fee = lender.flashFee(token, amount); uint256 _repayment = amount + _fee; IERC20(token).approve(address(lender), _allowance + _repayment); lender.flashLoan(this, token, amount, data); } }

```

Vat Dai

It may be that users are interested in moving dai around in the internal vat balances. Instead of wasting gas by minting/burning ERC20 dai you can instead use the vat dai flash mint function to short cut this.

The vat dai version of flash mint is roughly the same as the ERC20 dai version with a few caveats:

Function Signature

vatDaiFlashLoan(IVatDaiFlashBorrower receiver, uint256 amount, bytes calldata data)

vs

flashLoan(IERC3156FlashBorrower receiver, address token, uint256 amount, bytes calldata data)

Notice that no token is required because it is assumed to be vat dai. Also, theamount is inrad and not inwad .

Approval Mechanism

ERC3156 specifies using a token approval to approve the amount to repay to the lender. Unfortunately vat dai does not have a way to specify delegation amounts, so instead of giving the flash mint module full rights to withdraw any amount of vat dai we have instead opted to have the receiver push the balance owed at the end of the transaction.

Example

Here is an example similar to the one above to showcase the differences:

```

Copy pragma solidity ^0.6.12;

import "dss-interfaces/dss/VatAbstract.sol";

import "./interfaces/IERC3156FlashLender.sol"; import "./interfaces/IVatDaiFlashBorrower.sol";

contract FlashBorrower is IVatDaiFlashBorrower { enum Action {NORMAL, OTHER}

VatAbstract vat; IVatDaiFlashLender lender;

constructor ( VatAbstract vat_, IVatDaiFlashLender lender_ ) public { vat = vat_; lender = lender_; }

/// @dev Vat Dai Flash loan callback function onVatDaiFlashLoan( address initiator, uint256 amount, uint256 fee, bytes calldata data ) external override returns (bytes32) { require( msg.sender == address(lender), "FlashBorrower: Untrusted lender" ); require( initiator == address(this), "FlashBorrower: Untrusted loan initiator" ); (Action action) = abi.decode(data, (Action)); if (action == Action.NORMAL) { // do one thing } else if (action == Action.OTHER) { // do another }

// Repay the loan amount + fee // Be sure not to overpay as there are no safety guards for this vat.move(address(this), lender, amount + fee);

return keccak256("VatDaiFlashBorrower.onVatDaiFlashLoan"); }

/// @dev Initiate a flash loan function vatDaiFlashBorrow( uint256 amount ) public { bytes memory data = abi.encode(Action.NORMAL); lender.vatDaiFlashLoan(this, amount, data); } }

```

Previous DSR Manager - Detailed Documentation Next Maker Protocol Emergency Shutdown Last updated11 months ago

Export as PDF