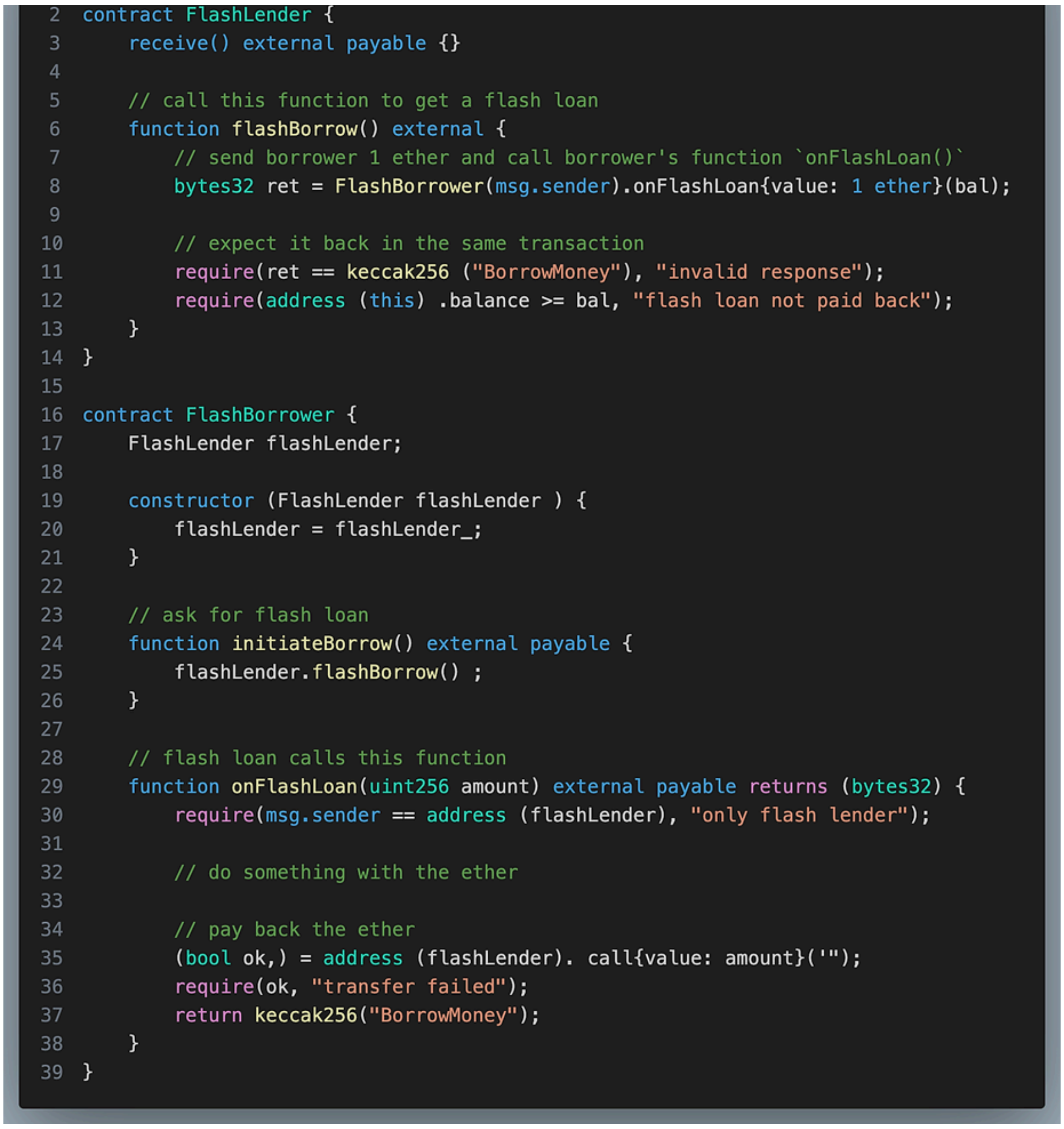
J Jeffrey Scholz Nov 7, 2023 8 min read

# Flash Loans and how to hack them: a walk through of ERC 3156

**Updated: Nov 18, 2023**

**Flash loans are loans between smart contracts that must be repaid in the same transaction. This article describes the ERC 3156 flash loan specification as well as the ways flash lenders and borrowers can be hacked. Suggested security exercises are provided at the end.**

**Below is an extremely simple example of a flash loan.**



```
contract FlashLender {
    receive() external payable {}

    // call this function to get a flash loan
    function flashBorrow() external {
        // send borrower 1 ether and call borrower's function `onFlashLoan()`
        bytes32 ret = FlashBorrower(msg.sender).onFlashLoan{value: 1 ether}(bal);

        // expect it back in the same transaction
        require(ret == keccak256 ("BorrowMoney"), "invalid response");
        require(address (this) .balance >= bal, "flash loan not paid back");
    }
}

contract FlashBorrower {
    FlashLender flashLender;

    constructor (FlashLender flashLender ) {
        flashLender = flashLender_;
    }

    // ask for flash loan
    function initiateBorrow() external payable {
        flashLender.flashBorrow() ;
    }

    // flash loan calls this function
    function onFlashLoan(uint256 amount) external payable returns (bytes32) {
        require(msg.sender == address (flashLender), "only flash lender");

        // do something with the ether

        // pay back the ether
        (bool ok,) = address (flashLender). call{value: amount}('"");
        require(ok, "transfer failed");
        return keccak256("BorrowMoney");
    }
}
```

If the borrower does not pay back the loan, the require statement with the message "flash not paid back" will cause the entire transaction to revert.

### Only contracts can work with flashloans

An EOA wallet cannot call a function to get the flash loan and then transfer the tokens back in a single transaction. Integration with a flash loan requires a separate smart contract.

### Flash loans do not need collateral

If a flash loan is implemented properly (big if!), then there is no risk of the loan not being paid back, because a revert or failed require statement will cause the transaction to fail, and the Ether will not transfer.

## What are flashloans used for?

profit. However, you need to money to buy the Ether in the first place. A flash loan is the ideal solution for it, as you don't need $1,200 lying around. You can borrow $1,200 of Ether, sell it for $1,300, and pay back the $1,200 keeping a $100 profit for yourself (minus fees).

## Refinancing Loans

For regular DeFi loans, they typically require some kind of collateral. For example, if you were borrowing $10,000 in stable coins, you would need to deposit $15,000 of Ether as collateral.

If your stable coins loan had a 5% interest and you wanted to refinance with another lending smart contract at 4%, you would need to

1. pay back the $10,000 in stable coins
2. withdraw the $15,000 Ether collateral
3. deposit the $15,000 Ether collateral into the other protocol
4. borrow $10,000 in stable coins again at the lower rate

This would be problematic if you had the $10,000 tied up in some other application. With a flashloan, you can do steps 1-4 without using any of your own stable coins.

## Exchanging collateral

In the example above, the borrower was using $15,000 of Ether as collateral. But suppose the protocol is offering a lower collateralization ratio using wBTC (wrapped bitcoin)? The borrower could use a flash loan and a similar set of steps outline above to swap out the collateral instead of the principal.

## Liquidating Borrowers

In the context of DeFi loans, if the collateral falls below a certain threshold, then the collateral can get liquidated — forcibly sold to cover the cost of the loan. In the example above, if the value of the Ether was to drop to $12,000, then the protocol might allow someone to purchase the Ether for $11,500 if they first pay back the $10,000 loan.

A liquidator could use a flash loan to pay off the $10,000 stable coin loan and receive $11,500. They would then sell this on another exchange for stable coins, and then pay back the flash loan.

## Increase yield for other DeFi applications

Uniswap and AAVE earn depositors' money through trading fees or lending interest. But since they have such a large amount of capital in one place, they can make additional money by also offering flash loans. This increases the efficiency of capital since the same capital now has more uses.

## Hacking Smart Contracts

Flash loans are probably most famous for their use by black hat hackers to exploit protocols. The primary attack vectors for flash loans are price manipulation and governance (vote) manipulation. Used on DeFi applications with inadequate defense, flash loans allow attackers to heavily buy up an asset increasing its price, or acquiring a bunch of voting tokens to push through a governance proposal.
The following is a list of flash loan hacks for the curious. Vulnerability is two-sided however. A flash lending and flash borrowing contract can also be vulnerable to losing money if not implemented properly.

## Examples of Flash Loan Hacks

Flash loan attacks are one of the most common exploits, presumably because developers coming from a web2 background aren't accustomed to it. Here are some of the more notorious examples.

rekt.news/deus-dao-rekt/
rekt.news/jimbo-rekt/
rekt.news/platypus-finance-rekt/
rekt.news/beanstalk-rekt/
rekt.news/inverse-rekt2/

Using flash loans to hack protocols is a separate topic, this article focuses on insecure implementations of flash lending and borrowing contracts.

implementation details need to be tied down, for example, should we call the function "getFlashLoan," "onFlashLoan," or something else? And then what parameters should it accept?

## ERC3156 Receiver Specification

The first aspect of the standard is the interface the borrower needs to implement, which is shown below. The borrower only needs to implement one function.

We describe the function arguments here

### initiator

This is the address that initiated the flash loan. **You probably want some kind of validation here so that untrusted addresses are not initiating flashloans on your contract.** Usually, the address would be ***you***, but you shouldn't assume that!

The function onFlashLoan is **expected** to be called by the flash loan contract, not the initiator. **You should check msg.sender is the flash loan contract inside the onFlashLoan() function because this function is external and anyone can call it.**

**Initiator is not msg.sender or the flash loan contract. It is the address that triggered the flash lending contract to call the receiver's onFlashLoan function.**

### token

This is the address of the ERC20 token you are borrowing. Contracts offering flash loans will usually hold several tokens they can flash loan out. The ERC3156 flash loan standard does not support flash loaning native Ether, but this can be implemented by flash loaning WETH and having the borrower unwrap the WETH. Because the borrowing contract is not necessarily the contract that called the flash loaner, the borrowing contract may need to be told what token is being flash lent.

### fee

Fee is how much of the token needs to be paid as a fee for the loan. It is denominated in absolute amount, not percentages.

### data

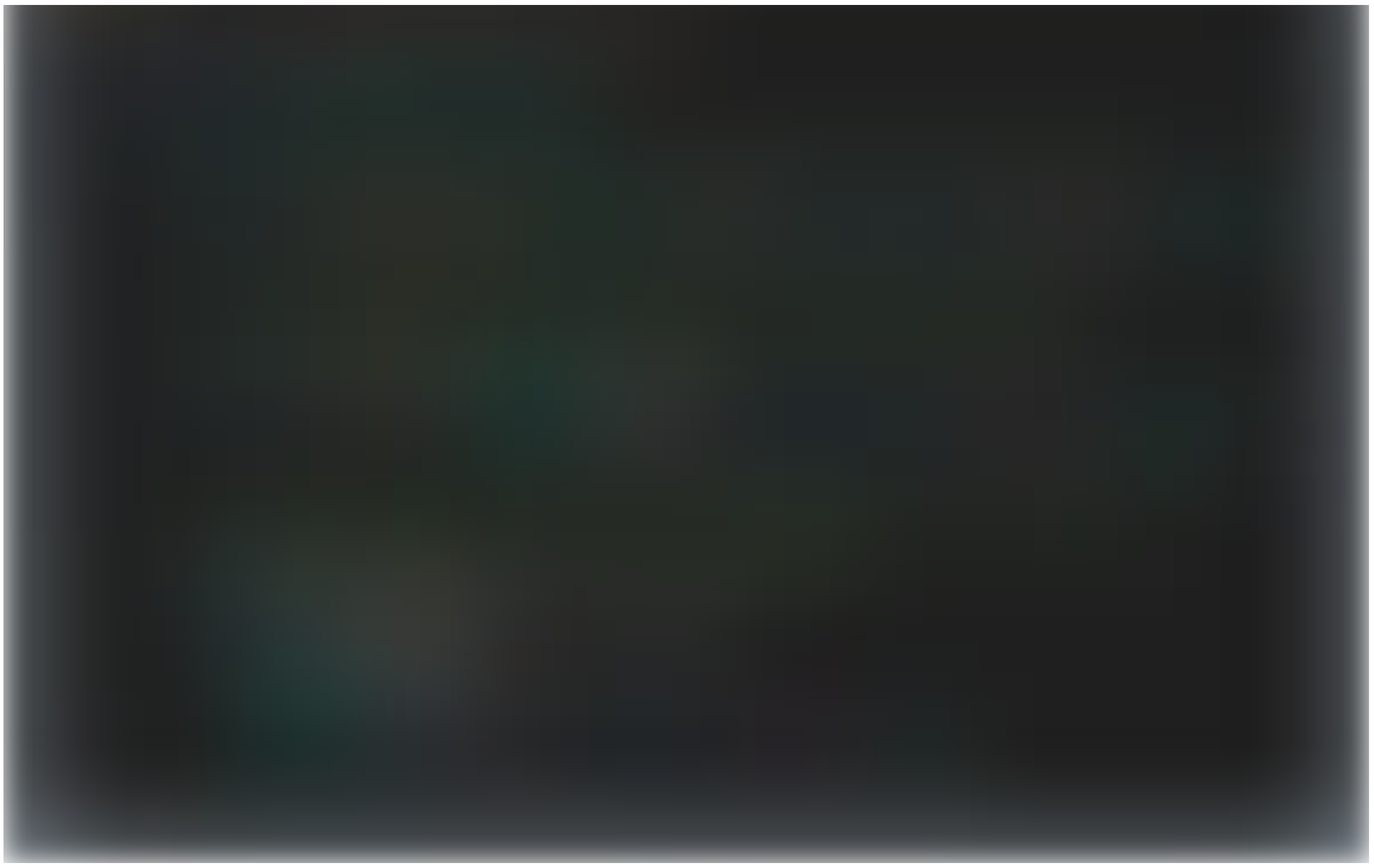
If your flash loan receiving contract isn't hard coded to take a particular action when receiving a flash loan, you can parameterize its behavior with the data parameter. For example, if your contract is arbitraging trading pools, then you would specify which pools to trade with.

### Reference implementation of the borrower

This has been modified from the code in the ERC 3156 spec to make the snippet smaller. Note that this contract is still placing perfect trust into the flash lender. **If the flash lender were somehow compromised, the contract below could be exploited through feeding it bogus amount and fee and initiator data.** If the lender is immutable, this isn't a concern, but it could be an attack vector if the lender is upgradeable.

## ERC3156 Lender Specification

Below is the interface for the lender specified by ERC3156

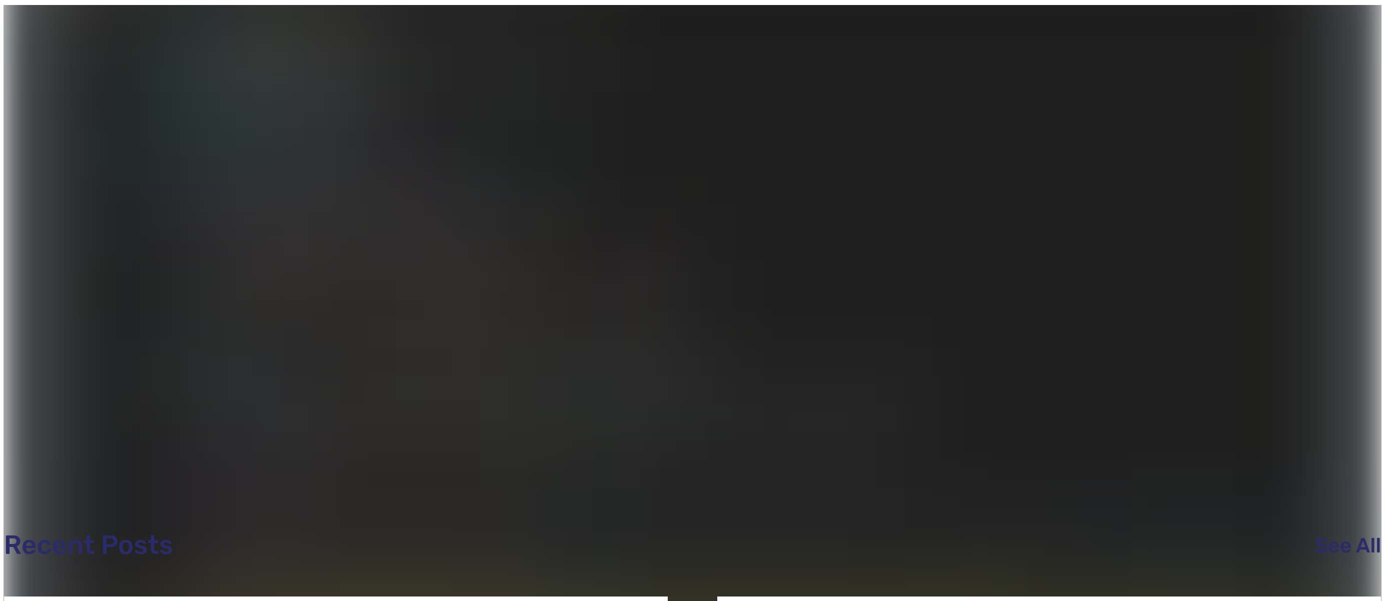


The arguments in the interface above have the same meaning as described in the previous section, so it won't be repeated here.

The flashLoan() function needs to accomplish a few important operations:

- Someone might call flashLoan() with a token the flash loan contract does not support. This should be checked for.
- Someone might call flashLoan() with an amount that is larger than maxFlashLoan. This also should be checked for
- data is simply forwarded to the caller.

More importantly, flashLoan() must transfer the tokens to the receiver ***and*** transfer them back. It should not rely on the borrower transferring the tokens back for repayment. The rational for this will be discussed in the next section. We have copied the reference implementation which can be found in the EIP 3156 Spec, here to emphasize the important parts:



Recent Posts

See All

C20 t

**Understanding the Function Selector in Solidity**

17 0 1

owe

only

**How ERC721 Enumerable Works**

176 0 2

onFlashLoan(). Otherwise, some actor other than the flash lender can call onFlashLoan() and cause unexpected behavior.

Furthermore, anyone can *call flashloan()* with an arbitrary borrower as the target and pass arbitrary data. To ensure the data is not

sees it's balance has returned to what it was before, but the borrower suddenly has become a lender with a large deposit.

UniswapV2's flash loan does not transfer the tokens back after the flash loan finishes. However, it uses a reentrancy lock to ensure that the borrower cannot "pay back the loan" by depositing it back into the protocol as if they were a lender.

## For the borrower, ensure only flash lender contract can call onFlashLoan

The flash lender is hardcoded to only call the receiver's onFlashLoan() function and nothing else. If a borrower had a way to specify which function the flash lender would call, then the flash loan could be manipulated into transferring other tokens in it's possession

## Practice Problems Related to Flash Loans

The following problems from DamnVulnerableDeFi and Mr Steal Yo Crypto can help you practice the attack vectors described above. One of the best way to understand flash loans is to learn **what not to do** when implementing them.

1. Naive Receiver (your goal is to drain the borrower, not necessarily steal their funds)
2. Side Entrance
3. Truster

Brush up on your knowledge of ERC 4626 and then practice these problems

1. Unstoppable (this one is a bit harder, so do it last. Your goal is to brick the contract, not steal the funds).
2. Flash Loaner (from Mr Steal Yo Crypto. Make sure you understand ERC 4626)

All of these problems are related to hacking the lender or the borrower, not using a flash loan to hack something else.

## Learn more with RareSkills

This material is part of our advanced Solidity Bootcamp. Please see the program to learn more.