```
    publi
    c {
    owne
    r =
    msg.s
    ende
    r;
  }

  function
    withdraw()
    public {
    require(owner
    ==
    msg.sender);
    msg.sender.transfer(address(this).balance);
  }

  function deposit(uint256 amount) public
    payable { require(msg.value ==
    amount);
  }

  function getBalance() public view
    returns (uint256) { return
    address(this).balance;
  }
}
```

I am going to generalize this contract to keep track of ether deposits based on the account address of the depositor, and then only allow that same account to make withdrawals of that ether. To do this, we need a way keep track of account balances for each depositing account—a mapping from accounts to balances. Fortunately, Solidity provides a ready-made mapping data type that can map account addresses to integers,

The code above demonstrates the following:

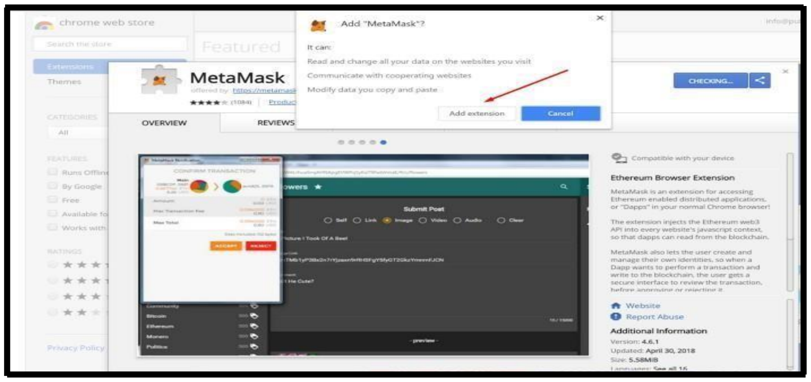
- The require(amount <= balances[msg.sender]) checks to make sure the sender has sufficient funds to cover the requested withdrawal. If not, then the transaction aborts without making any state changes or ether transfers.
- The balanceOf mapping must be updated to reflect the lowered residual amount after the withdrawal.
- The funds must be sent to the sender requesting the withdrawal.

In the withdraw() function above, it is very important to adjust balanceOf[msg.sender] **before** transferring ether to avoid an exploitable vulnerability. The reason is specific to smart contracts and the fact that a transfer to a smart contract executes code in that smart contract. (The essentials of Ethereum transactions are discussed in How Ethereum Transactions Work.)

Now, suppose that the code in withdraw() did not adjust balanceOf[msg.sender] before making the
transfer *and* suppose that msg.sender was a malicious smart contract. Upon receiving the transfer—handled by msg.sender's fallback function—that malicious contract could initiate

another withdrawal from the banking contract. When the banking contract handles this second withdrawal request, it would have already transferred ether for the original withdrawal, but it would not have an updated balance, so it would allow this second withdrawal!

To avoid this sort of reentrancy bug, follow the "Checks-Effects-Interactions pattern" as described in the Solidity documentation. The withdraw()function above is an example of implementing this pattern



Add "MetaMask"?

It can:

Read and change all your data on the websites you visit

Communicate with cooperating websites

Modify data you copy and paste

Ethereum Browser Extension

MetaMask is an extension for accessing Ethereum enabled distributed applications, or "Dapps" in your normal Chrome browser!

The extension injects the Ethereum web3 API into every website's javascript context, so that dapps can read from the blockchain.

MetaMask also lets the user create and manage their own identities, so when a Dapp wants to perform a transaction and write to the blockchain, the user gets a secure interface to review the transaction, before approving or rejecting it.

Additional Information

Version: 4.6.1

Updated: April 30, 2018

Size: 5.58MiB