Signature Malleability

Description

The implementation of a cryptographic signature system in Ethereum contracts often assumes that the signature is unique, but signatures can be altered without the possession of the private key and still be valid. The EVM specification defines several so-called ‘precompiled’ contracts one of them being ecrecover which executes the elliptic curve public key recovery. A malicious user can slightly modify the three values *v*, *r* and *s* to create other valid signatures. A system that performs signature verification on contract level might be susceptible to attacks if the signature is part of the signed message hash. Valid signatures could be created by a malicious user to replay previously signed messages.

Remediation

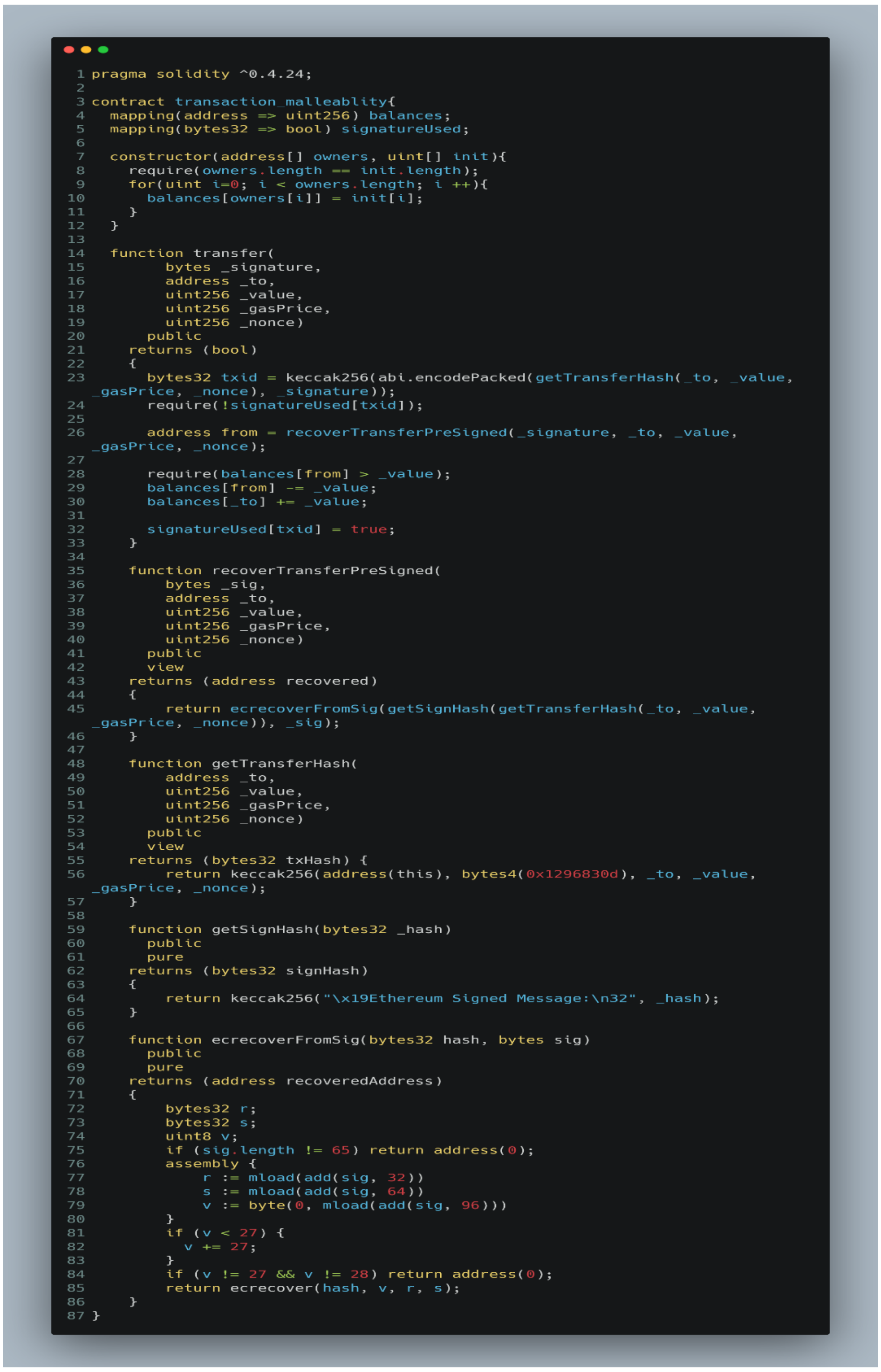
A signature should never be included into a signed message hash to check if previously messages have been processed by the contract.

Example:

Code Explanation:

Don’t assume that the use of a cryptographic signature system in smart contracts verifies that signatures are unique, however, this isn't the case. Signatures in Ethereum can be altered without the private key and remain valid. For example, elliptic key cryptography consists of three variables: v, r, and s and if these values are modified in just the right way, you can obtain a valid signature with an invalid private key.

Code:



```solidity
pragma solidity ^0.4.24;

contract transaction_malleablity{
  mapping(address => uint256) balances;
  mapping(bytes32 => bool) signatureUsed;

  constructor(address[] owners, uint[] init){
    require(owners.length == init.length);
    for(uint i=0; i < owners.length; i ++){
      balances[owners[i]] = init[i];
    }
  }

  function transfer(
        bytes _signature,
        address _to,
        uint256 _value,
        uint256 _gasPrice,
        uint256 _nonce)
      public
    returns (bool)
    {
      bytes32 txid = keccak256(abi.encodePacked(getTransferHash(_to, _value, _gasPrice, _nonce), _signature));
      require(!signatureUsed[txid]);

      address from = recoverTransferPreSigned(_signature, _to, _value, _gasPrice, _nonce);

      require(balances[from] > _value);
      balances[from] -= _value;
      balances[_to] += _value;

      signatureUsed[txid] = true;
    }

    function recoverTransferPreSigned(
        bytes _sig,
        address _to,
        uint256 _value,
        uint256 _gasPrice,
        uint256 _nonce)
      public
      view
    returns (address recovered)
    {
        return ecrecoverFromSig(getSignHash(getTransferHash(_to, _value, _gasPrice, _nonce)), _sig);
    }

    function getTransferHash(
        address _to,
        uint256 _value,
        uint256 _gasPrice,
        uint256 _nonce)
      public
      view
    returns (bytes32 txHash) {
        return keccak256(address(this), bytes4(0x1296830d), _to, _value, _gasPrice, _nonce);
    }

    function getSignHash(bytes32 _hash)
      public
      pure
    returns (bytes32 signHash)
    {
        return keccak256("\x19Ethereum Signed Message:\n32", _hash);
    }

    function ecrecoverFromSig(bytes32 hash, bytes sig)
      public
      pure
    returns (address recoveredAddress)
    {
        bytes32 r;
        bytes32 s;
        uint8 v;
        if (sig.length != 65) return address(0);
        assembly {
            r := mload(add(sig, 32))
            s := mload(add(sig, 64))
            v := byte(0, mload(add(sig, 96)))
        }
        if (v < 27) {
          v += 27;
        }
        if (v != 27 && v != 28) return address(0);
        return ecrecover(hash, v, r, s);
    }
}
```