Parity Wallet Hacks: Postmortem

Zihan Zheng
Jerry Chen
Ethan Wang
Jakub Jackowiak

Overview

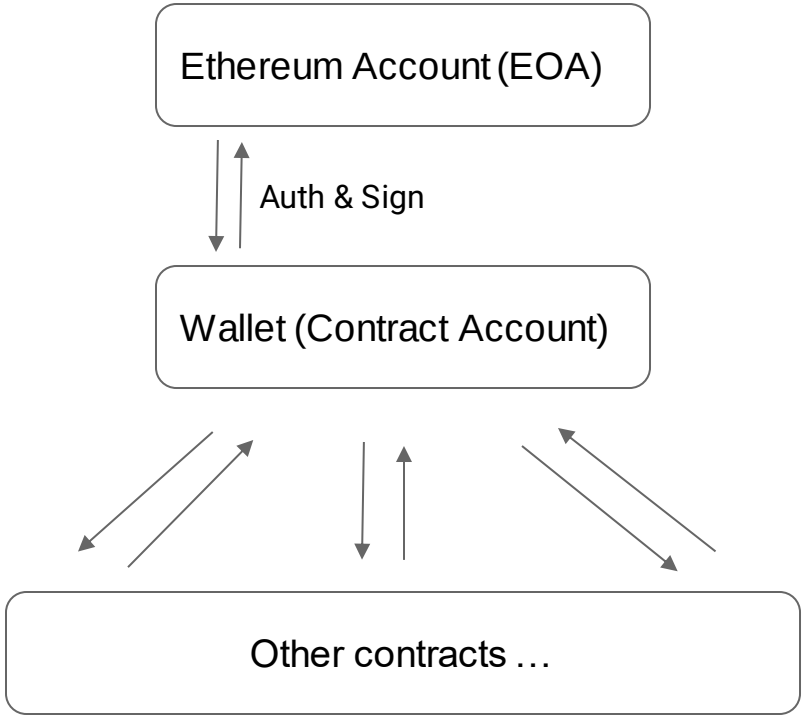
1. Wallet Account Explained
2. Implementation Choices
3. Postmortem:

    The First Hack: July 19, 2017

    The Second Hack: Nov 6, 2017

1. Challenges Explained

Wallet Account Explained



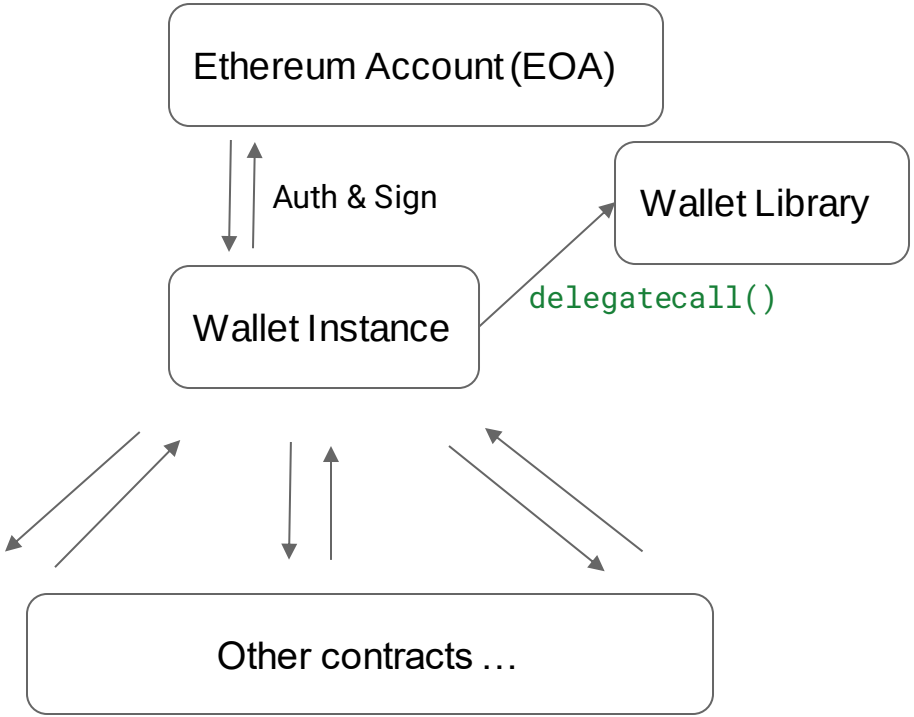
A wallet is essentially a product for managing the ethereum account(s).

Wallets can provide additional features:

- Multisig: another layer of security/robustness
- Quotas and Limits: More detailed management
- Ownership Transfer, Logging, etc.

Additional feature means additional code, as well as additional gas and risks!

Implementation Choice



A wallet library contains function implementations for the wallet contract, and is deployed separately.

Benefits:

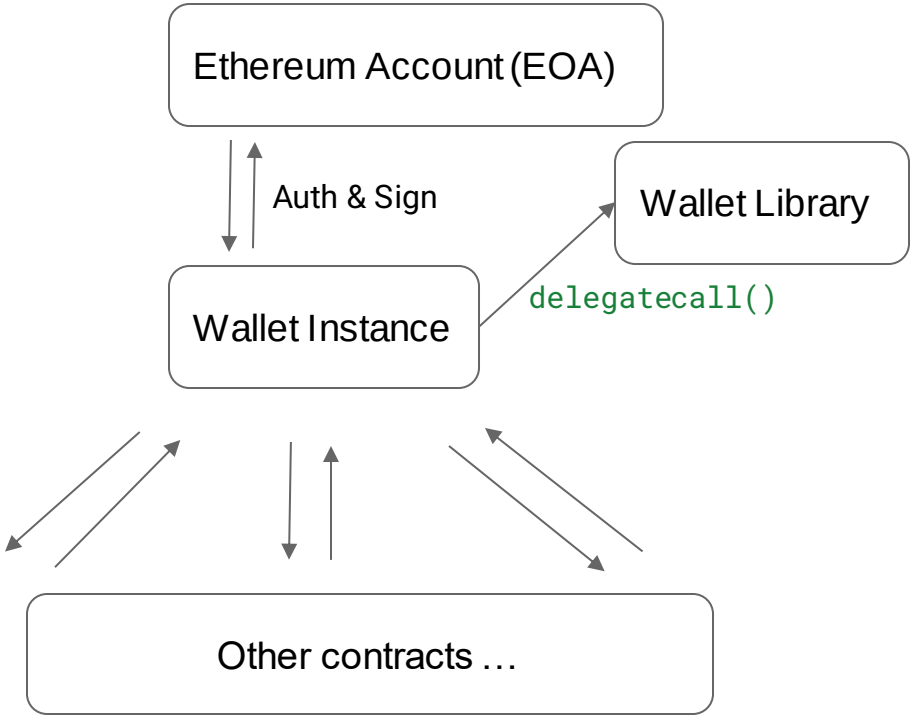
- Save expensive storage
- Reduction of gas for wallet deployment
- Upgradability of the library functions

  * function updates

  * security updates

  without changing the wallet instance

Implementation Choice



A wallet library contains function implementations for the wallet contract, and is deployed separately.

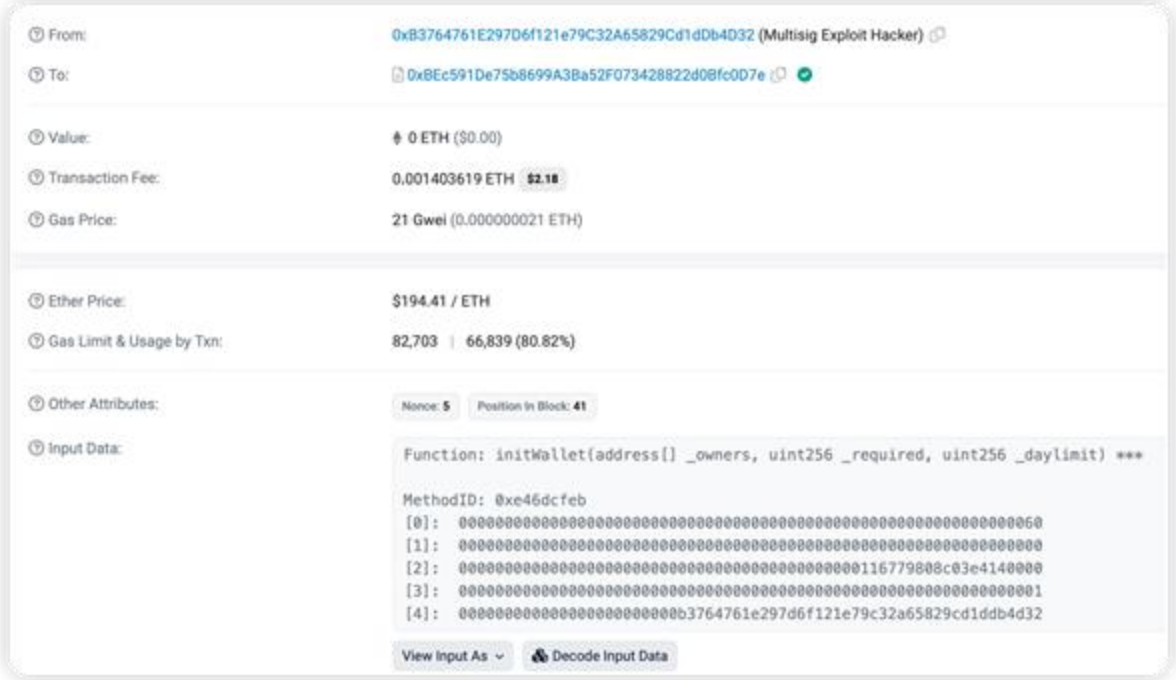
Risks:

- Delegatecall can modify states.
- Additional maintenance for the library.

The First Hack: July 19, 2017: Ownership Claimed

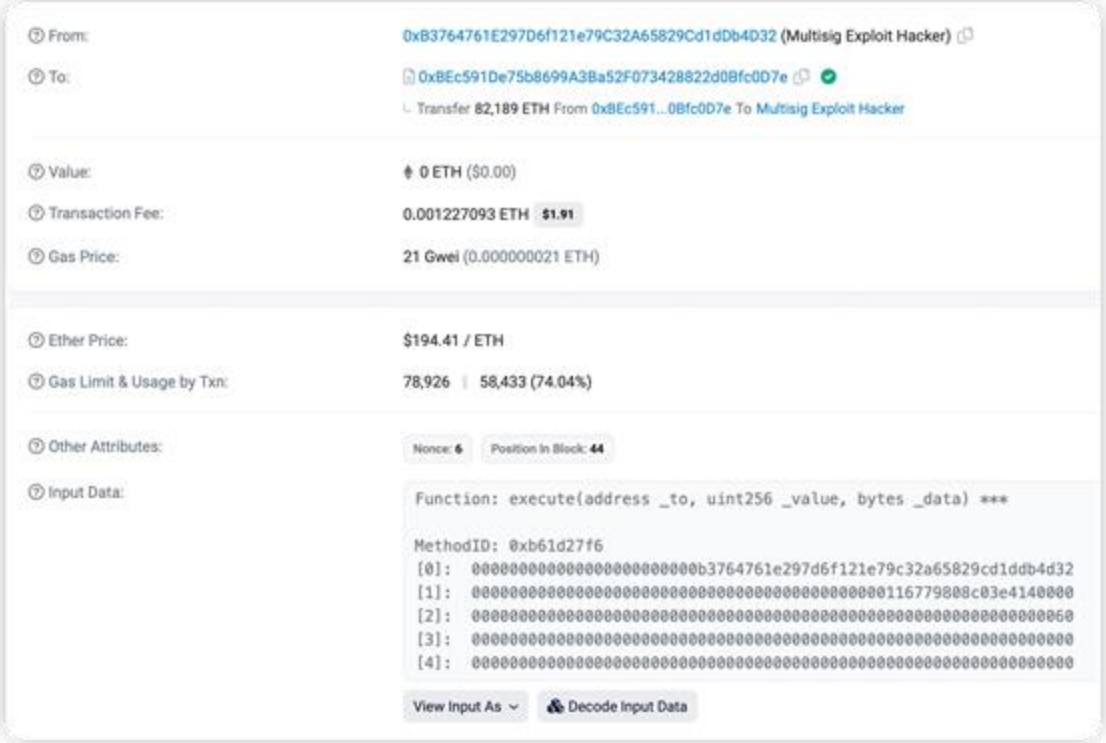
Wallet contracts were compromised.

The attacker claim sole ownership of the user's multisig wallet contract using `delegatecall()` in the fallback function.



The First Hack: July 19, 2017: Balance Transferred

Then the attacker issued a transaction draining all the remaining funds from the wallet.



The First Hack: July 19, 2017: Vulnerable Code

Code Snippet of the enhanced-wallet.sol:

```
214     // constructor - just pass on the owner array to the multiowned and
215     // the limit to daylimit
216     function initWallet(address[] _owners, uint _required, uint _daylimit) {
217       initDaylimit(_daylimit);
218       initMultiowned(_owners, _required);
219     }
220
221     // kills the contract sending everything to `_to`.
222     function kill(address _to) onlymanyowners(sha3(msg.data)) external {
223       suicide(_to);
224     }
```

External?

Internal?

Private?

Public?

The First Hack: July 19, 2017: Vulnerable Code

Code Snippet of the enhanced-wallet.sol:

```
// constructor is given number of sigs required to do protected
// as well as the selection of addresses capable of confirming
function initMultiowned(address[] _owners, uint _required) {
  m_numOwners = _owners.length + 1;
  m_owners[1] = uint(msg.sender);
  m_ownerIndex[uint(msg.sender)] = 1;
  for (uint i = 0; i < _owners.length; ++i)
  {
    m_owners[2 + i] = uint(_owners[i]);
    m_ownerIndex[uint(_owners[i])] = 2 + i;
  }
  m_required = _required;
}
```

External?

Internal?

Private?

Public?

The First Hack: July 19, 2017: the Fix

```
105  105      // constructor is given number of sigs required to do protected "onlymanyowners" transactions
106  106      // as well as the selection of addresses capable of confirming them.
107       -   function initMultiowned(address[] _owners, uint _required) {
     107  +   function initMultiowned(address[] _owners, uint _required) internal {
108  108        m_numOwners = _owners.length + 1;
109  109        m_owners[1] = uint(msg.sender);
110  110        m_ownerIndex[uint(msg.sender)] = 1;
```

```
216       -   function initWallet(address[] _owners, uint _required, uint _daylimit) {
     219  +   function initWallet(address[] _owners, uint _required, uint _daylimit) only_uninitialized {
217  220        initDaylimit(_daylimit);
218  221        initMultiowned(_owners, _required);
219  222      }
```

```
+   modifier only_uninitialized { if (m_numOwners > 0) throw; _; }
```

Will this fix work?

The First Hack: July 19, 2017: Another Fix

```
function initMultiowned(address[] _owners, uint _required) only_uninitialized {
  m_numOwners = _owners.length + 1;
  m_owners[1] = uint(msg.sender);
  m_ownerIndex[uint(msg.sender)] = 1;
```

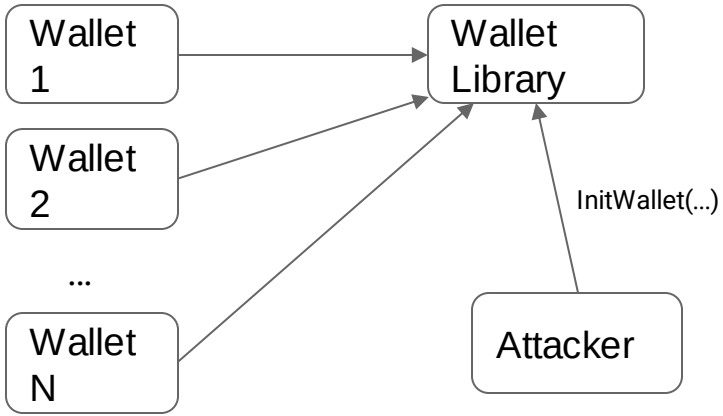
```
216     -   function initWallet(address[] _owners, uint _required, uint _daylimit) {
        219 +   function initWallet(address[] _owners, uint _required, uint _daylimit) only_uninitialized {
217     220         initDaylimit(_daylimit);
218     221         initMultiowned(_owners, _required);
219     222       }
```

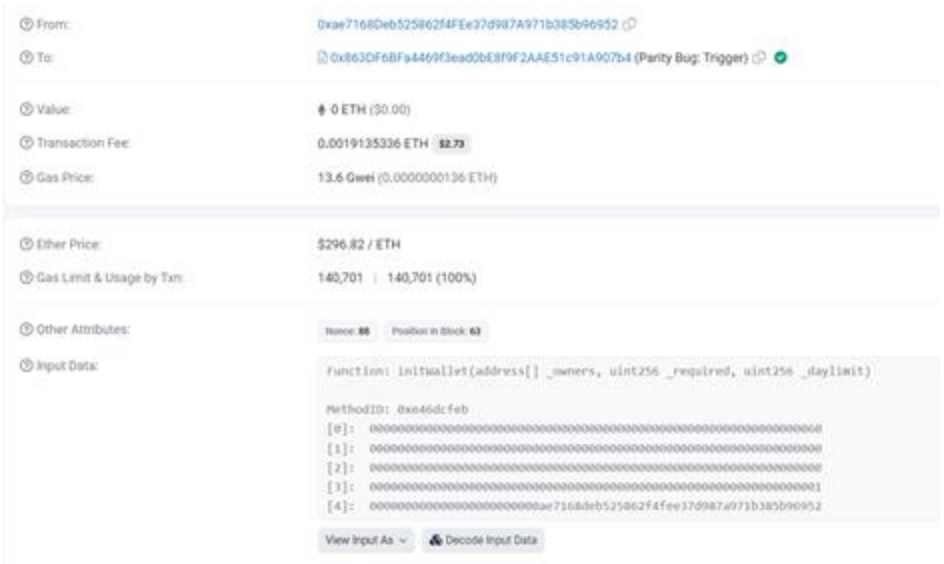
```
+   modifier only_uninitialized { if (m_numOwners > 0) throw; _; }
```

Will this fix work?

The Second Hack: Nov 6, 2017

- Function initWallet hadn't been called and owner had not been initialized on the library contract
- User calls `initWallet()` on the library contract function and becomes owner





Call to `initWallet()` on the library contract

The Second Hack: Nov 6, 2017

As owner, attacker calls the self-destruct function making all Parity multi-sig wallet funds unrecoverable

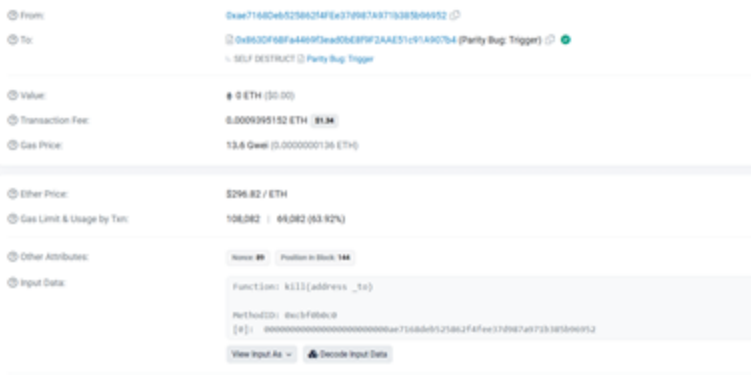
Two main points of failure:

- Inclusion of self-destruct function, which remained from the original contract
- Failure to initialize the library contract wallet
  - Recommended by a user three months prior, but didn't consider possibility for contract to self-destruct

Any subsequent calls to library functions return false

```
221    // kills the contract sending everything to `_to`.
222    function kill(address _to) onlymanyowners(sha3(msg.data)) external
223      suicide(_to);
224    }
```

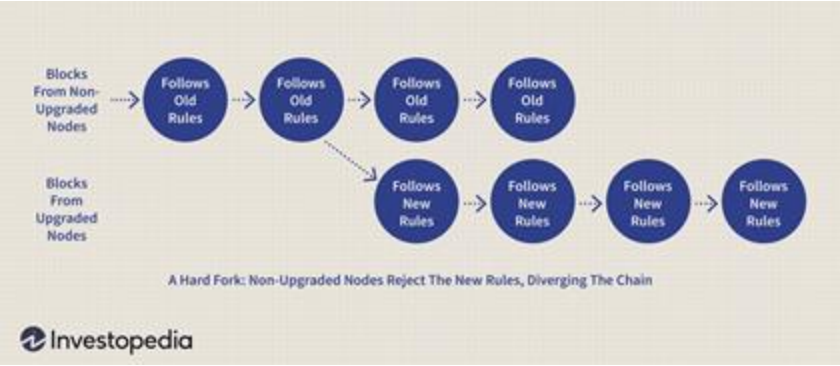
Function self-destructs contract and sends all funds to the address in argument



Call to self-destruct function

Aftermath of Second Hack: Nov 6, 2017

- 513,774.16 ETH ($115 million) of funds frozen from over 587 wallets
- No clear way to unfreeze the funds
  - Hard Fork required
  - EIP156
  - EIP999
- Funds still frozen due to disagreements



Parity took the following actions

- Paused deployments of multisig wallets
- Security Audits
- Establishing procedures for code review and contract deployments
- Extending research and bug bounty programs

Lessons Learned: at the cost of 150k ETH(~$30M)

- Balancing security and savings
- Bugs are a likelihood, even with trivial code
- Code should be reviewed by external auditors in addition to peer-review
- More community auditing
- Define guards and specify function visibility in the code
- Solidity can have improved security design
- Limit copying/pasting of Defi contracts
    - Aftermath of attack: Funds in other wallets with same vulnerability were recovered by a white hat group
- Decrease contract complexity

Lessons Learned: at the cost of 514k ETH(~$115M)

- Use `library` instead of `contract` keyword
- `selfdestruct` in libraries is risky
- Libraries introduce single point of failure
- Correctly initialize library contracts
- Fully review and consider security implications of found issues

Wallet Contract Design:

- Keep complexity minimal
- Follow Solidity best practices
- Conduct end-to-end testing prior to deployment

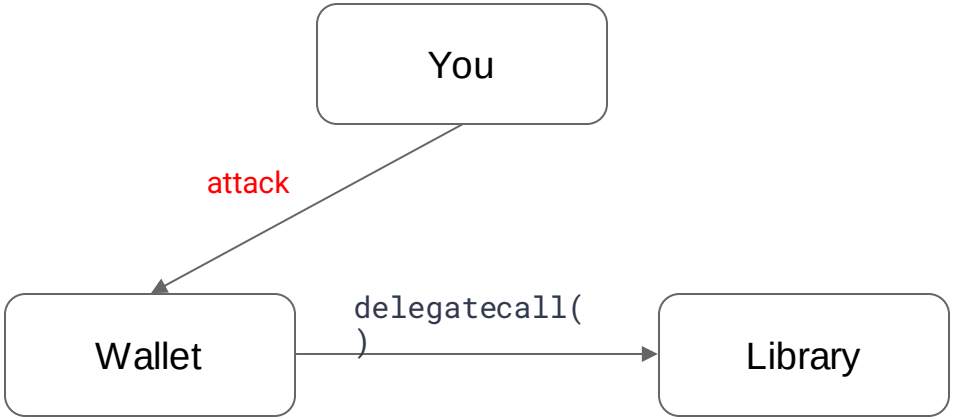
Refactoring as a library implementation:

- Stateless
- Link wallet contracts with Solidity `using` keyword on a data struct
- Unable to self destruct
- Avoids catch-all delegate calls and makes clear which functions are called from contract

Challenges Explained: Hack the Wallet!

Simplified WalletLibrary and Wallet contracts.

Claim ownership of the wallet!



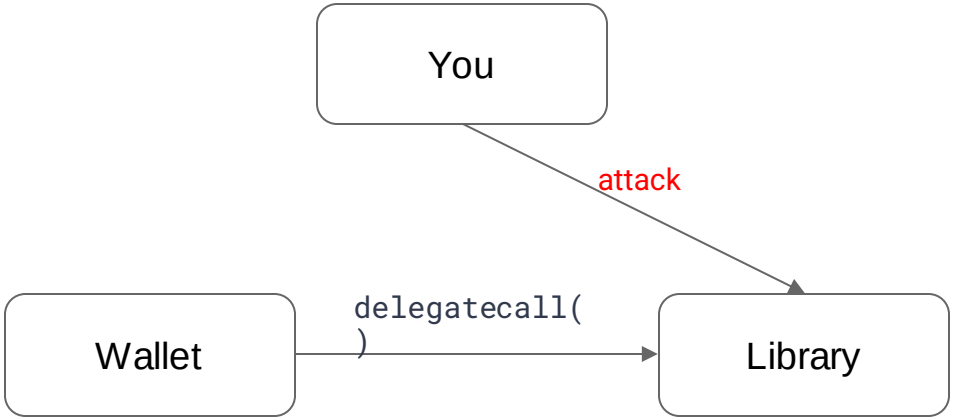

```
contract WalletLibrary {
  address public walletLibAddr;
  ...
  function initWallet(address payable _owner) external {
        owner = _owner;
    }
  ...
}
```

```
contract Wallet {
  address public walletLibAddr;
  ...
  fallback() external payable {
        (bool success /* bytes memory returnedData */, ) = walletLibAddr
            .delegatecall(msg.data);
        require(success, "[delegatecall] failed");
    }
    ...
}
```

Challenges Explained: Hack the Library!

Simplified WalletLibrary and Wallet contracts.

devops199 “I accidentally killed it”

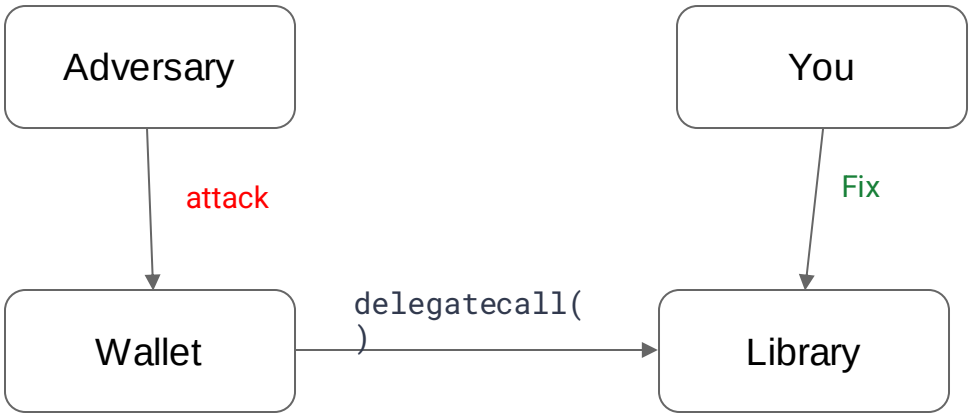



```
contract WalletLibrary {
  address public walletLibAddr;
  ...
  function initWallet(address payable _owner) external {
        owner = _owner;
    }
  ...
}
```

```
contract Wallet {
  address public walletLibAddr;
  ...
  fallback() external payable {
        (bool success /* bytes memory returnedData */, ) = walletLibAddr
            .delegatecall(msg.data);
        require(success, "[delegatecall] failed");
    }
    ...
}
```

Challenges Explained: the Fix

Implement a WalletLibrary contract that's safe against the previous two attacks.




```
function tryAttack() public {
    if (player_lib == address(0)) {
        return;
    }
    Wallet _wallet = new Wallet(payable(player_lib));
    testWallet = address(_wallet);
    attackTried = true;
    (bool success, bytes memory data) = address(_wallet).call(
        abi.encodeWithSignature("initWallet(address)", address(0))
    );
    // (bool success, ) = .delegatecall(abi.encodeWithSignature());
}

function completed() external returns (bool) {
    tryAttack();
    if (attackTried == true && testWallet != address(0)) {
        Wallet _testWallet = Wallet(payable(testWallet));
        if (_testWallet.owner() != address(0)) {
            return true;
        }
```