zeppelin-solidity Documentation

Release 1.0.0

Zeppelin

Aug 16, 2017

Contents

Zeppelin is a library for writing secure Smart Contracts on Ethereum.

With Zeppelin, you can build distributed applications, protocols and organizations:

- using *Common Contract Security Patterns*
- in the Solidity language.

The code is open-source, and available on github.

CHAPTER 1

Getting Started

Zeppelin integrates with Truffle, an Ethereum development environment. Please install Truffle and initialize your project with `truffle init`:

```
npm install -g truffle
mkdir myproject && cd myproject
truffle init
```

To install the Zeppelin library, run:

```
npm i zeppelin-solidity
```

After that, you'll get all the library's contracts in the contracts/zeppelin folder. You can use the contracts in the library like so:

```
import "zeppelin-solidity/contracts/Ownable.sol";

contract MyContract is Ownable {
  ...
}
```

> NOTE: The current distribution channel is npm, which is not ideal. We're looking into providing a better tool for code distribution , and ideas are welcome.

Truffle Beta Support

We also support Truffle Beta npm integration. If you're using Truffle Beta, the contracts in `node_modules` will be enough, so feel free to delete the copies at your `contracts` folder. If you're using Truffle Beta, you can use Zeppelin contracts like so:

```
import "zeppelin-solidity/contracts/Ownable.sol";

contract MyContract is Ownable {
```

```
  ...
}
```

For more info see the Truffle Beta package management tutorial.

CHAPTER 2

Ownable

Base contract with an owner.

Ownable( )

Sets the address of the creator of the contract as the owner.

modifier onlyOwner( )

Prevents function from running if it is called by anyone other than the owner.

transferOwnership(address newOwner) onlyOwner

Transfers ownership of the contract to the passed address.

CHAPTER 3

Claimable

Extension for the Ownable contract, where the ownership needs to be claimed

transfer(address newOwner) onlyOwner

Sets the passed address as the pending owner.

modifier onlyPendingOwner

Function only runs if called by pending owner.

claimOwnership( ) onlyPendingOwner

Completes transfer of ownership by setting pending owner as the new owner.

CHAPTER 4

Migrations

Base contract that allows for a new instance of itself to be created at a different address.

Inherits from contract Ownable.

upgrade(address new_address) onlyOwner

Creates a new instance of the contract at the passed address.

setCompleted(uint completed) onlyOwner**

Sets the last time that a migration was completed.

CHAPTER 5

SafeMath

Provides functions of mathematical operations with safety checks.

assert(bool assertion) internal

Throws an error if the passed result is false. Used in this contract by checking mathematical expressions.

mul(uint256 a, uint256 b) internal returns (uint256)

Multiplies two unsigned integers. Asserts that dividing the product by the non-zero multiplicand results in the multiplier.

sub(uint256 a, uint256 b) internal returns (uint256)

Checks that b is not greater than a before subtracting.

add(uint256 a, uint256 b) internal returns (uint256)

Checks that the result is greater than both a and b.

CHAPTER 6

LimitBalance

Base contract that provides mechanism for limiting the amount of funds a contract can hold.

LimitBalance(unit _limit)

Constructor takes an unisgned integer and sets it as the limit of funds this contract can hold.

modifier limitedPayable()

Throws an error if this contract's balance is already above the limit.

CHAPTER 7

PullPayment

Base contract supporting async send for pull payments. Inherit from this contract and use asyncSend instead of send.

asyncSend(address dest, uint amount) internal

Adds sent amount to available balance that payee can pull from this contract, called by payer.

withdrawPayments( )

Sends designated balance to payee calling the contract. Throws error if designated balance is 0, if contract does not hold enough funds ot pay the payee, or if the send transaction is not successful.

CHAPTER 8

StandardToken

Based on code by FirstBlood: Link FirstBloodToken.sol

Inherits from contract SafeMath. Implementation of abstract contract ERC20 (see https://github.com/ethereum/EIPs/issues/20)

approve(address _spender, uint _value) returns (bool success)

Sets the amount of the sender's token balance that the passed address is approved to use.

allowance(address _owner, address _spender) constant returns (uint remaining)

Returns the approved amount of the owner's balance that the spender can use.

balanceOf(address _owner) constant returns (uint balance)

Returns the token balance of the passed address.

transferFrom(address _from, address _to, uint _value) returns (bool success)

Transfers tokens from an account that the sender is approved to transfer from. Amount must not be greater than the approved amount or the account's balance.

function transfer(address _to, uint _value) returns (bool success)

Transfers tokens from sender's account. Amount must not be greater than sender's balance.

CHAPTER 9

BasicToken

Simpler version of StandardToken, with no allowances

balanceOf(address _owner) constant returns (uint balance)

Returns the token balance of the passed address.

function transfer(address _to, uint _value) returns (bool success)

Transfers tokens from sender's account. Amount must not be greater than sender's balance.

CHAPTER 10

Bounty

To create a bounty for your contract, inherit from the base *Bounty* contract and provide an implementation for `deployContract()` returning the new contract address.:

```
import {Bounty, Target} from "./zeppelin/Bounty.sol";
import "./YourContract.sol";

contract YourBounty is Bounty {
function deployContract() internal returns(address) {
return new YourContract()
  }
}
```

Next, implement invariant logic into your smart contract. Your main contract should inherit from the Target class and implement the checkInvariant method. This is a function that should check everything your contract assumes to be true all the time. If this function returns false, it means your contract was broken in some way and is in an inconsistent state. This is what security researchers will try to acomplish when trying to get the bounty.

At contracts/YourContract.sol:

```
import {Bounty, Target} from "./zeppelin/Bounty.sol";
contract YourContract is Target {
  function checkInvariant() returns(bool) {
    // Implement your logic to make sure that none of the invariants are broken.
  }
}
```

Next, deploy your bounty contract along with your main contract to the network.

At `migrations/2_deploy_contracts.js`:

```
module.exports = function(deployer) {
  deployer.deploy(YourContract);
  deployer.deploy(YourBounty);
};
```

Next, add a reward to the bounty contract

After deploying the contract, send reward funds into the bounty contract.

From `truffle console`:

```
bounty = YourBounty.deployed();
address = 0xb9f68f96cde3b895cc9f6b14b856081b41cb96f1; // your account address
reward = 5; // reward to pay to a researcher who breaks your contract

web3.eth.sendTransaction({
  from: address,
  to: bounty.address,
  value: web3.toWei(reward, "ether")
})
```

If researchers break the contract, they can claim their reward.

For each researcher who wants to hack the contract and claims the reward, refer to our Test for the detail.

Finally, if you manage to protect your contract from security researchers, you can reclaim the bounty funds. To end the bounty, destroy the contract so that all the rewards go back to the owner.:

```
bounty.destroy();
```

CHAPTER 11

Common Contract Security Patterns

Zeppelin smart contracts are developed using industry standard contract security patterns and best practices. To learn more, please see Onward with Ethereum Smart Contract Security.

CHAPTER 12

Developer Resources

Building a distributed application, protocol or organization with Zeppelin?

Ask for help and follow progress at: https://zeppelin-slackin.herokuapp.com/

Interested in contributing to Zeppelin?

- Framework proposal and roadmap: https://medium.com/zeppelin-blog/zeppelin-framework-proposal-and-development-roadmap-fdfa9a3a32ab#.iain47pak
- Issue tracker: https://github.com/OpenZeppelin/zeppelin-solidity/issues
- Contribution guidelines: https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/CONTRIBUTING.md

CHAPTER 13

The MIT License (MIT)