

---

# **Zilliqa Token Audit**

Zero Knowledge Labs Auditing Services

AUTHOR: MATTHEW DI FERRANTE

2017-01-07

## Audited Material Summary

The audit consists of the following contracts:

```
1 ded84c783b29ff583a39d49750da2bf96696788ff06c89d907c4477713007290
    ZilliqaToken.sol
```

The contract implements a ERC20 Token, with [Pausable](#) functionality. ERC20 and [Pausable](#) implementation are taken from Zeppelin's solidity code. It also allows token holders to burn their tokens, reducing the supply.

## Security

There are no security issues in the code.

### ZilliqaToken.sol

The [ZilliqaToken](#) contract inherits from Zeppelin's [PausableToken](#):

```
1 contract ZilliqaToken is PausableToken
```

The contract has only one custom modifier, [validDestination](#), which ensures that the argument address is not zero nor the contract's address.

## Constructor

```
1 function ZilliqaToken( address _admin, uint _totalTokenAmount )
2 {
3     // assign the admin account
4     admin = _admin;
5
6     // assign the total tokens to zilliqa
7     totalSupply = _totalTokenAmount;
8     balances[msg.sender] = _totalTokenAmount;
9     Transfer(address(0x0), msg.sender, _totalTokenAmount);
10 }
```

The constructor sets the contract's [admin](#), [totalSupply](#), and emits a [Transfer](#) event notifying a token creation event from [0x0](#) to [msg.sender](#).

## transfer

```
1   function transfer(address _to, uint _value) validDestination(_to)
2       returns (bool)
3   {
4       return super.transfer(_to, _value);
5   }
```

The `transfer` function overrides the standard ERC20 `transfer` to apply the `validDestination` modifier. All else remains the same.

## transferFrom

```
1   function transferFrom(address _from, address _to, uint _value)
2       validDestination(_to) returns (bool)
3   {
4       return super.transferFrom(_from, _to, _value);
5   }
```

The `transferFrom` function overrides the standard ERC20 `transferFrom` to apply the `validDestination` modifier. Like for `transfer`, all else **remains** the same.

## burn

```
1   function burn(uint _value) returns (bool)
2   {
3       balances[msg.sender] = balances[msg.sender].sub(_value);
4       totalSupply = totalSupply.sub(_value);
5       Burn(msg.sender, _value);
6       Transfer(msg.sender, address(0x0), _value);
7       return true;
8   }
```

The `burn` function allows a token holder to destroy their coins, reducing the total supply. SafeMath is used when manipulating balances so there are no arithmetic security issues.

On **succes**, a `Burn` event is emitted, as well as a `Transfer` event notifying a token transfer of `_value` from `msg.sender` to `0x0`.

## burnFrom

```
1   function burnFrom(address _from, uint256 _value) returns (bool)
2   {
3       assert( transferFrom( _from, msg.sender, _value ) );
4       return burn(_value);
5   }
```

The `burnFrom` function is a helper that allows smart contracts calling the token contract to burn tokens in one call.

## emergencyERC20Drain

```
1   function emergencyERC20Drain( ERC20 token, uint amount ) onlyOwner {
2       // owner can drain tokens that are sent here by mistake
3       token.transfer( owner, amount );
4   }
```

This function allows the contract owner to claim and `rescue` arbitrary ERC20 tokens sent to this contract by mistake.

## **Disclaimer**

This audit concerns only the correctness of the Smart Contracts listed, and is not to be taken as an endorsement of the platform, team, or company.

## **Audit Attestation**

This audit has been signed by the key provided on <https://keybase.io/mattdf> - and the signature is available on <https://github.com/mattdf/audits/>