# Checks for integer overflow
There is a class SafeMath on line #3. Class SafeMath defines methods that protect from integer overflow as well as defines the method `assert`. There is a Class Token on line #28, derived from SafeMath that only uses the method `assert`. Methods that protect from integer overflow are not used at all. Instead, checks against integer overflow are duplicated all over the file. It is recommended to rework such pieces of code:

```
if((balances[msg.sender] < _value) || (balances[_to] + _value <=
balances[_to])) {
  throw;
}
balances[msg.sender] -= _value;
balances[_to] += _value;
```

to

```
require(balances[msg.sender] < _value);
balances[msg.sender] = safeSub(balances[msg.sender], _value);
balances[_to] = safeAdd(balances[_to], _value);
```

This will follow common Solidity code practices, make code more readable and reduce the risk of a bug.

# Redundant implementations of methods
Class `Token` on line #28 defines ERC20 methods `totalSupply`, `balanceOf`, `transfer` etc. All these methods have empty implementations (Note the empty implementation in curly brackets):

`function totalSupply() constant returns (uint256 supply) {}`

Contract `StdToken` on line #65 is derived from `Token` and contains actual implementations of those methods. It is recommended to make the class `Token` abstract and remove empty implementations, i.e.:

`function totalSupply() constant returns (uint256 supply);`

This will simplify the lookup method, make bytecode of compiled contracts smaller and make transactions a little bit cheaper.

There is a quite simple inheritance problem:

- You define the method totalSupply() (line 27)
- In the derived contract you define the public variable totalSupply (line 65). Because this is a public variable - compiler auto-create public getter totalSupply(). This auto-created getter overrides your own method totalSupply() from line 27 and everything works.
- When you delete implementation from line 27 – the compiler does not create the auto-generated getter. Instead it makes contract abstract (because you need to define your own implementation of totalSupply()) and after that the contract cannot be deployed to

the blockchain. The error is quite obvious: The contract code couldn't be stored, please check your gas amount.

- To fix all of this quickly (I tested it now, this works):
    - On line 27, paste the content of line 65: `uint public totalSupply = 0;`
    - Delete line 65

# Security issues

Method `approve` of a contract `StdToken` on the line #101 is vulnerable to the known security bug described here: https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
It does not affect ICO, but impacts the usage of the token after the ICO finished.

# Redundant code

The modifier `onlyPayloadSize` of the contract `StdToken` on line #112 is defined but not used. It is recommended to delete it to reduce the contract’s size.

# Contract `StdToken`

The contract `StdToken` implements functionality of standard ERC20 token.
No special features at all. There is a very well-known library for smart contracts development - `zeppelin-solidity` by ConsenSys. This library contains the contract `StandardToken` that implements the same features.
`StandardToken` is very well implemented and very well tested. No need to re-invent the wheel.
It is recommended to replace `StdToken` with the `StandardToken` by `zeppelin-solidity`.
At least, this will give more trust from the community. Also, if the contract `StdToken` will be replaced with `StandardToken` - it solves all issues mentioned above.

# Parameters of `EthLendToken`

Constructor of a contract `EthLendToken` takes several parameters: `_tokenManager`, `_escrow`, `_teamTokenBonus`
It is recommended to hardcode all these params in the smart contract because of two reasons:
1. It will help avoid mistakes during the deployment of a contract to production.
2. Etherscan is the main Ethereum block explorer. This web-site has a feature to show source code of deployed contracts. Sometimes this feature does not work for contracts that have a parametrized constructor (usually works as expected).

In order to make unit tests it is possible to make a test contract `EthLendTokenTest` with the parametrized constructor:

```
`contract EthLendTokenTest is EthLendToken {...}`
```

# Code style
Contracts use the special word `throw` which is deprecated. It is recommended to use `require`/`assert` instead.
The latest versions of `solc` compiler produce error messages for `throw`.
Common recommendations for development of smart contracts - do not use contracts in production mode that have errors during compilation.

# Version of minimum compiler version
Currently, the minimum version of `solc` compiler is specified as `0.4.4`: `pragma solidity ^0.4.4;`
It is recommended to upgrade it to at least `0.4.13` because of many security improvements in compiler since `0.4.4`.

# Specification of a contract
I do not have full specs of the contract so I cannot check prices/bonuses/etc.
Contract conforms content of the doc:
https://docs.google.com/document/d/14TXp1bOB1eDFygpIRDjam2BKbadLwuIa1HPZAFClwuc

# Code style of unit tests
It is recommended to apply `eslint` to the source code of unit tests.
It is also recommended to use async/await in unit tests (instead of callbacks).
This will greatly increase readability of unit tests.

# Custom development environment
To test the contract - custom development environment was used.
`npm test` does not work out of the box.
Git repo does not contain instructions on how to reproduce this environment and launch tests.
As a result, I was not able to launch unit tests.

It looks like we need to manually setup test Ethereum node, activate needed accounts etc.
Currently de-facto standard for development of smart contracts - `truffle` framework.
`Truffle` automate all that computational work.

# Conclusion
One security issue found. It does not affect ICO, but it needs to be fixed for later use of the token.

Code style and used development environment does not follow the best practices for development of smart contracts.

It looks like ICO contracts should work properly, but I was not able to launch unit tests and check it.

Audit brief

The audit of the ETHLend ICO smart contract has been performed in Aug-Sept 2017.
Last reviewed git commit - 9c789ea
https://github.com/ETHLend/ICO_SmartContract

The audit revealed several technical issues:

- mess with integer overflow checks
- not the best inheritance patterns
- custom Solidity code instead of well-reviewed code of public libraries (for example, Zeppelin-Solidity)
- use of old Solidity compilers (missing security features introduced in new versions of Solidity)
- controversial design pattern of contract initialization
- code style issues
- miscellaneous minor issues

All these issues did not affect the security of the ICO contract directly.
But at the same time these issues reduced the readability of the code and created an opportunity for a human error.

Most of the issues are fixed.

One of the not fixed issues - design pattern of contract initialization.
With the current implementation, the contract receives several parameters during the deployment phase. It is possible to make mistakes regarding the values of these parameters. Therefore, it is required to check 3 parameters of a deployed contract before the ICO launch:

- tokenManager
- teamTokenBonus
- escrow

Conclusion:

The ICO contract is safe for use in production.

Victor Mezrin

https://github.com/mezrin

mezrinv@gmail.com