**AVOIDING COMMON ATTACKS**

To check for the security of the contract, I look at different type of attacks in order to make sure that the contract is secure and follows best practices.

**REENTRANCY ATTACK**

This attack consists on recursively calling the call.value () method in a ERC20 token to extract the ether stored on the contract if the user is not updating the balance of the sender before sending the ether.

When you call a function to send ether to a contract, you can use the fallback function to execute again that function until the ether of the contract is extracted.

Because this contract does not use call.value (), there’s no risk of reentrancy attacks since the transfer function only allows to use 23.000 gas which you can only use for an event to log data and throws on failure.

That way you’re unable to recursively call again the sender function thus avoiding the reentrancy attack.

**OVER AND UNDER FLOWS**

An overflow happens when the limit of type variable uint256, 2\*\*256, is exceeded. What happens is that the value resets to zero instead of incrementing more.

For example, if I want to assign a value to a uint bigger than 2\*\*256 it will simply go to 0 – this is dangerous.

On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0.

For example, if you subtract 0 – 1 the result will be = 2\*\*256 instead of -1.

OpenZepplin’s SafeMath.sol will also protect against integer over and underflows.

The way you use it is by importing the library, activating, it for uint256 and then using the function .mul(), .add(), sub() and .div().

**REPLAY ATTACK**

The replay attack consists on making a transaction on the blockchain like the original Ethereum’s blockchain and then repeating it on another blockchain like the Ethereum classic blockchain.

This contract is safe from a Replay attack

**DESIGN PATTERNS CONSIDERED WHILE CREATING THIS DAPP**

Some of the best practices used to develop this code:

Using audited and battle tested code

Calling external contracts

Modular code

Access Restriction:

Secure Ether Transfer:

Emergency Stop:

Proxy Delegate:

Contract Self Destruction