// HALBORN

BlackHole Protocol

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **July 2nd, 2021 - July 7th, 2021**

Visit: Halborn.com

Table of Content

• Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

• Overview of the audit

The project has 1 file. It contains approx 2201 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation, but that does not create any vulnerability.

• Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

- ## Over and under flows

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

This contract **does** check for overflows and underflows by using OpenZeppelin's SafeMath to mitigate this attack, but all the functions have strong validations, which prevented this attack.

- ## Short address attack

If the token contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the ethereum's virtual machine will just add zeros to the transaction until the address is complete.

Although this contract **is not vulnerable** to this attack, but there are some point where users can mess themselves due to this (Please see below). It is highly recommended to call functions after checking validity of the address.

- ## Visibility & Delegate call

It is also known as, The Parity Hack, which occurs while misuse of Delegate call.

No such issues found in this smart contract and visibility also properly addressed. There are some places where there is no visibility defined. Smart Contract will assume "Public" visibility if there is no visibility defined. It is good practice to explicitly define the visibility, but again, the contract is not prone to any vulnerability due to this in this case.

- ## Reentrancy / TheDAO hack

Reentrancy occurs in this case: any interaction from a contract (A) with another contract (B) and any transfer of ethereum hands over control to that contract (B).

This makes it possible for B to call back into A before this interaction is completed.

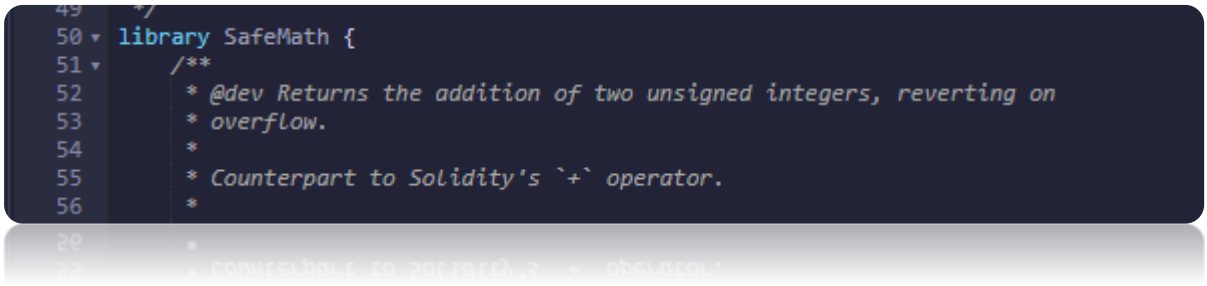
Use of "require" function in this smart contract mitigated this vulnerability.

- **Forcing Ethereum to a contract**

While implementing "selfdestruct" in smart contract, it sends all the ethereum to the target address. Now, if the target address is a contract address, then the fallback function of target contract does not get called. And thus Hacker can bypass the "Required" conditions. Here, the Smart Contract's balance has never been used as guard, which mitigated this vulnerability.

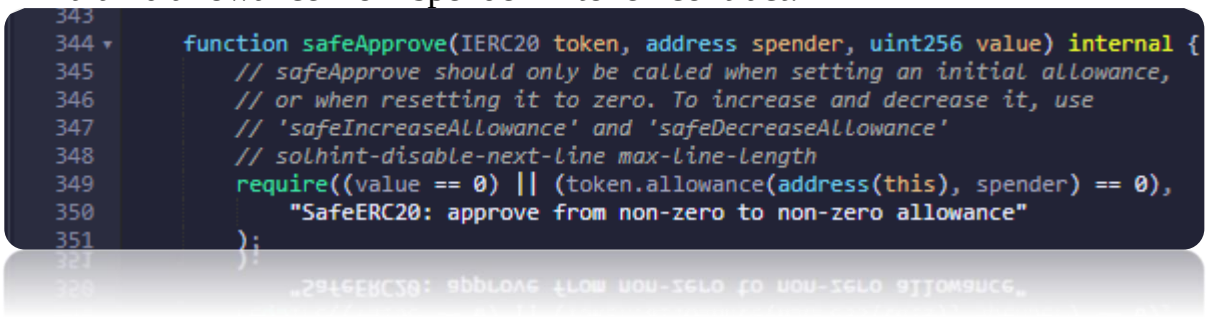
Good things in smart contract

- **SafeMath library:-**
  - You are using SafeMath library it is a good thing. This protects you from underflow and overflow attacks.



```
*/
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
```

- **Good required condition in functions:-**
  - Here you are checking that value should be not 0, and contract address has more than 0 allowance from spender in token contract.



```
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
```

- Here you are checking that token address is contract address, transfer to token address is successfully done.

```
370      */
371      function _callOptionalReturn(IERC20 token, bytes memory data) private {
372          // We need to perform a low level call here, to bypass Solidity's return d
373          // we're implementing it ourselves.
374
375          // A Solidity high level call has three parts:
376          //  1. The target address is checked to verify it contains contract code
377          //  2. The call itself is made, and success asserted
```

- Here you are checking that newAdmin address value should be valid and proper.

```
580    function changeAdmin(address newAdmin) external ifAdmin {
581      require(newAdmin != address(0), "Cannot change the admin of a proxy to the zer
582      emit AdminChanged(_admin(), newAdmin);
583      _setAdmin(newAdmin);
```

```
773    function __changeAdmin__(address newAdmin) external ifAdmin {
774      require(newAdmin != address(0), "Cannot change the admin of a proxy to the zer
775      emit AdminChanged(_admin(), newAdmin);
776      _setAdmin(newAdmin);
777    }
```

- Here you are checking that _implementation should return not address value.

```
910    function initialize(address _logic, address _admin, bytes memory _data) public p
911      require(_implementation() == address(0));
912      InitializableUpgradeabilityProxy.initialize(_logic, _data);
913      assert(ADMIN_SLOT == bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1));
914      _setAdmin(_admin);
915    }
```

- Here you are checking that newFactory address should be contract address.

```
953    function _setFactory(address newFactory) internal {
954      require(OpenZeppelinUpgradesAddress.isContract(newFactory), "Cannot set a fact
955
956      bytes32 slot = FACTORY_SLOT;
```

- Here you are checking that factory should return not address value.

```
1003   function initialize(address factory, bytes memory data) public payable {
1004     require(_factory() == address(0));
1005     assert(FACTORY_SLOT == bytes32(uint256(keccak256('eip1967.proxy.factory')) -
1006     _setFactory(factory);
```

- Here you are checking that contract is already been initialized.

```
1072     */
1073   modifier initializer() {
1074     require(initializing || isConstructor() || !initialized, "Contract instance
1075
1076     bool isTopLevelCall = !initializing;
```

- Here you are checking that newGovernor address value should be valid and proper.

```
1222    function _transferGovernorship(address newGovernor) internal {
1223        require(newGovernor != address(0));
1224        emit GovernorshipTransferred(governor, newGovernor);
1225        governor = newGovernor;
1226    }
```

- Here you are checking that amount value should be more than 0. You have to add amount value to totalSupply after transfer is successfully done.

```
1382    function stakeWithPermit(uint256 amount, uint deadline, uint8 v, bytes32 r, b
1383        require(amount > 0, "Cannot stake 0");
1384        _totalSupply = _totalSupply.add(amount);
1385        balances[msg.sender] = balances[msg.sender].add(amount);
```

```
1394    function stake(uint256 amount) virtual override public nonReentrant updateRew
1395        require(amount > 0, "Cannot stake 0");
1396        _totalSupply = _totalSupply.add(amount);
1397        balances[msg.sender] = balances[msg.sender].add(amount);
```

- Here you are checking that amount value should be more than 0. You have to add amount value to totalSupply after transfer is successfully done. You need to check that user has more or equal balance to amount.

```
1402    function withdraw(uint256 amount) virtual override public nonReentrant update
1403        require(amount > 0, "Cannot withdraw 0");
1404        _totalSupply = _totalSupply.sub(amount);
1405        _balances[msg.sender] = _balances[msg.sender].sub(amount);
```

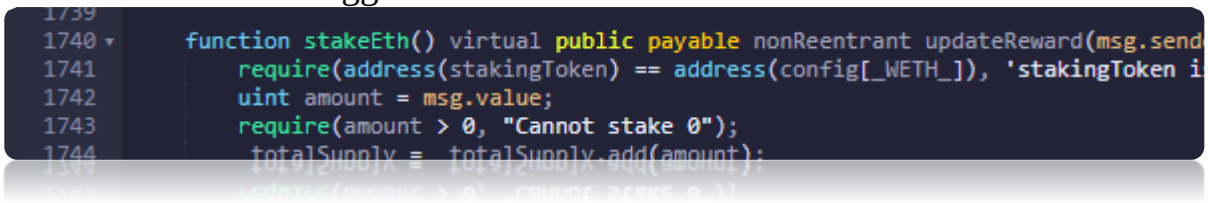
- Here you are checking that rewardRate should be not too high.

```
1438        // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
1439        uint balance = rewardsToken.balanceOf(address(this));
1440        require(rewardRate <= balance.div(rewardsDuration), "Provided reward too
1441
```

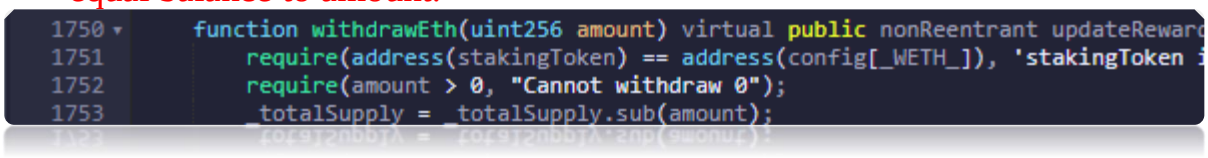
- Here you are checking that acct value should not be in blocklist, and checking that acct value should not be contract.

```
1593    function getRewardA(address payable acct) virtual public nonReentrant updateR
1594        require(getConfigA(_blocklist_, acct) == 0, 'In blocklist');
1595        bool isContract = acct.isContract();
1596        require(!isContract || config[_allowContract_] != 0 || getConfigA(_allowl
1597
```

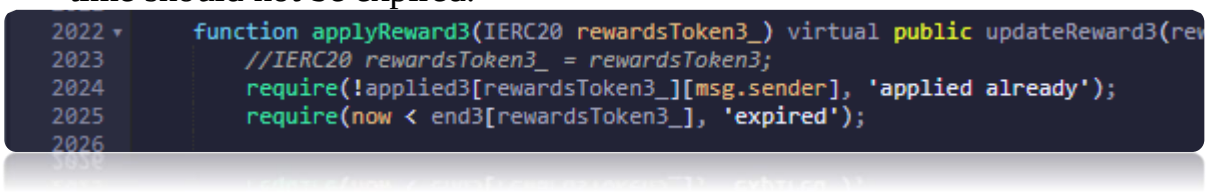
- Here you are checking that stakingToken should not be WETH, and amount value should be bigger than 0.



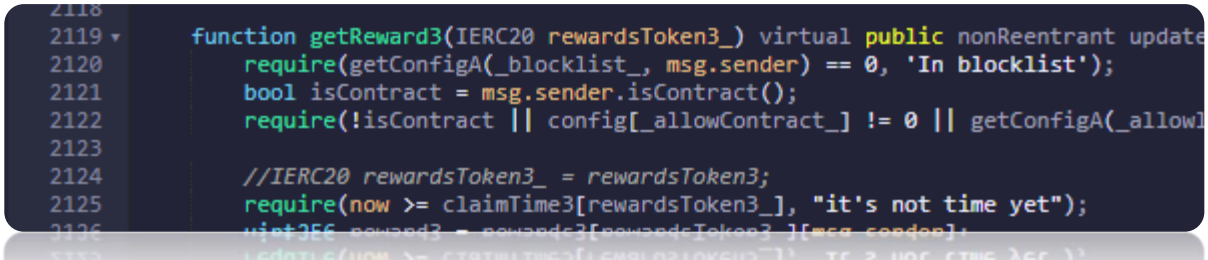
- Here you are checking that stakingToken should not be WETH, and amount value should be bigger than 0. You need to check that msg.sender has more or equal balance to amount.



- Here you are checking that Reward should not be applied already, and Reward time should not be expired.



- Here you are checking that msg.sender should not be in blocklist and msg.sender is not contract value and current time should be bigger than claim time for reward.



• Critical vulnerabilities found in the contract

=> No Critical vulnerabilities found

• Medium vulnerabilities found in the contract

=> No Medium vulnerabilities found

- # Low severity vulnerabilities found

  - ## 7.1: Uncheck return response of transfer method:-

=> I have found that you are transferring fund to address using a transfer method.

=> It is always good to check the return value or response from a function call.

=> Here are some functions where you forgot to check a response.

=> I suggest, if there is a possibility then please check the response.

- **Function: - withdrawEth**

```
1750    function withdrawEth(uint256 amount) virtual public nonReentrant updateReward
1751        require(address(stakingToken) == address(config[_WETH_]), 'stakingToken i
1752        require(amount > 0, "Cannot withdraw 0");
1753        _totalSupply = _totalSupply.sub(amount);
1754        _balances[msg.sender] = _balances[msg.sender].sub(amount);
1755        IWETH(address(stakingToken)).withdraw(amount);
1756        msg.sender.transfer(amount);
1757        emit Withdrawn(msg.sender, amount);
```

  - Here you are calling transfer method 1 time. It is good to check that the transfer is successfully done or not.

  - ## 7.2: Compiler version is not fixed:-

=> In this file you have put “pragma solidity ^0.6.0;” which is not a good way to define compiler version.

=> Solidity source files indicate the versions of the compiler they can be compiled with. Pragma solidity >=0.6.0; // bad: compiles 0.6.0 and above pragma solidity 0.6.0; //good: compiles 0.6.0 only

=> If you put(>=) symbol then you are able to get compiler version 0.6.0 and above. But if you don’t use(^/>=) symbol then you are able to use only 0.6.0 version. And if there are some changes come in the compiler and you use the old version then some issues may come at deploy time.

=> Use latest version of solidity.

• Summary of the Audit

Overall the code is well and performs well. There is no back

door to steal fund.

Please try to check the address and value of token externally before sending to the solidity code.

Our final recommendation would be to pay more attention to the visibility of the functions , hardcoded address and mapping since it’s quite important to define who’s supposed to executed the functions and to follow best practices regarding the use of assert, require etc. (which you are doing ;)).

- **Good Point:** Code performance is good. Code is written in secure way. All address and value validations done properly.
- **Suggestions:** Please try to check return response of transfer method, use latest and static version of solidity, and check user address value in every function.

THANK YOU FOR CHOOSING

// HALBORN