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 2 files. It contains approx 2534 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
- • **Filename: - ZOMHEALTH.sol**
- • **SafeMath library (For all files):-**
  - o You are using SafeMath library it is a good thing. This protects you from underflow and overflow attacks.

```
8
9
10 ▾  library SafeMath {
11
12 ▾      function add(uint256 a, uint256 b) inte
13             uint256 c = a + b;
14             require(c >= a, "SafeMath: addition
15
16             return c;
```

- • **Good required condition in functions:-**
  - o Here you are checking that newOwner address value is proper.

```
187      */
188 ▾     function transferOwnership(address newOwner) public onlyOw
189            require(newOwner != address(0), "Ownable: new owner is
190            emit OwnershipTransferred(_owner, newOwner);
191            _owner = newOwner;
192        }
```

  - o Here you are checking that sender and recipient addresses are proper.

```
521      */
522 ▾     function _transfer(address sender, address recipient, uint2
523            require(sender != address(0), "ERC20: transfer from the
524            require(recipient != address(0), "ERC20: transfer to th
525
526            _beforeTokenTransfer(sender, recipient, amount);
527            _balances[sender] = _balances[sender].sub(amount, "ERC2
```

o Here you are checking that a account address value is proper.

```
540          */
541 ▾    function _mint(address account, uint256 amount) intern
542            require(account != address(0), "ERC20: mint to the
543
544            _beforeTokenTransfer(address(0), account, amount)
545
546            _totalSupply = _totalSupply.add(amount);
```

o Here you are checking that an account address value is proper.

```
561          */
562 ▾    function _burn(address account, uint256 amount) internal vir
563            require(account != address(0), "ERC20: burn from the zer
564
565            _beforeTokenTransfer(account, address(0), amount);
566
567            balances[account] = balances[account] sub(amount "ERC
```

o Here you are checking that an owner and spender addresses value is proper.

```
583      * -   spender   cannot be the zero address.
584          */
585 ▾    function _approve(address owner, address spender, uint256 am
586            require(owner != address(0), "ERC20: approve from the ze
587            require(spender != address(0), "ERC20: approve to the ze
588
589            _allowances[owner][spender] = amount;
590            emit Approval(owner, spender, amount);
```

o Here you are checking that signatory is proper address, nonce value is correct as per coming from outside, expiry time is bigger than current time.

```
805          address signatory = ecrecover(digest, v, r, s);
806          require(signatory != address(0), "ZOM::delegateBySig: invalid signature");
807          require(nonce == nonces[signatory]++, "ZOM::delegateBySig: invalid nonce");
808          require(now <= expiry, "ZOM::delegateBySig: signature expired");
809          return _delegate(signatory, delegatee);
810      }
```

o Here you are checking that an blockNumber is less than current block number.

```
833      function getPriorVotes(address account, uint blockNumber)
834          external
835          view
836          returns (uint256)
837 ▾    {
838          require(blockNumber < block.number, "ZOM::getPriorVotes
839
```

- **FileName: - ZOM_OBSIDIAN.sol**

- **Good required condition in functions:-**

o Here you are checking that a newOwner address is peoper.

```
77 ▾     function transferOwnership(address newOwner) public onlyOwner {
78           require(newOwner != address(0), "Ownable: new owner is the ze
79           emit OwnershipTransferred(_owner, newOwner);
80           _owner = newOwner;
81       }
82  }
```

o Here you are checking that an index value is smaller than length of set value.

```
210        */
211 ▾     function _at(Set storage set, uint256 index) private view retu
212            require(set._values.length > index, "EnumerableSet: index
213            return set._values[index];
214        }
215
```

o Here you are checking that an amount balance is less than or equal contract balance and call function is successfully called.

```
374        */
375 ▾     function sendValue(address payable recipient, uint256 amount) internal {
376            require(address(this).balance >= amount, "Address: insufficient balance");
377
378            // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
379            (bool success, ) = recipient.call{ value: amount }("");
380            require(success, "Address: unable to send value, recipient may have reverte
381        }
382
```

o Here you are checking that a value is balance of contract is bigger than value.

```
434        * _Available since v3.1._
435        */
436 ▾     function functionCallWithValue(address target, bytes memory data,
437            require(address(this).balance >= value, "Address: insufficient
438            return _functionCallWithValue(target, data, value, errorMessag
439        }
```

o Here you are checking that a target address is proper.

```
439        }
440
441 ▾     function _functionCallWithValue(address target, bytes memory data, uint256 we
442            require(isContract(target), "Address: call to non-contract");
443
444            // solhint-disable-next-line avoid-low-level-calls
445            (bool success, bytes memory returndata) = target.call{ value: weiValue }(
446 ▾         if (success) {
```

o Here you are checking that a sender and recipient addresses are proper.

```
935        */
936 ▾     function _transfer(address sender, address recipient, uint
937            require(sender != address(0), "ERC20: transfer from th
938            require(recipient != address(0), "ERC20: transfer to t
939
940            _beforeTokenTransfer(sender, recipient, amount);
```

o Here you are checking that an account address is proper.

```
954        */
955 ▾     function _mint(address account, uint256 amount) internal virtu
956            require(account != address(0), "ERC20: mint to the zero ac
957
958            _beforeTokenTransfer(address(0), account, amount);
959
```

- Here you are checking that an account address is proper.

```
975         */
976 ▾      function _burn(address account, uint256 amount) internal
977             require(account != address(0), "ERC20: burn from the
978
979             _beforeTokenTransfer(account, address(0), amount);
980
```

- Here you are checking that an owner and spender addresses are proper.

```
998         */
999 ▾      function _approve(address owner, address spender, uint256 amount) internal
1000            require(owner != address(0), "ERC20: approve from the zero address");
1001            require(spender != address(0), "ERC20: approve to the zero address");
1002
1003            _allowances[owner][spender] = amount;
```

- Here you are checking that signatory is proper address, nonce value is correct as per coming from outside, expiry time is bigger than current time.

```
1220        address signatory = ecrecover(digest, v, r, s);
1221        require(signatory != address(0), "ZOM::delegateBySig: invalid signature");
1222        require(nonce == nonces[signatory]++, "ZOM::delegateBySig: invalid nonce");
1223        require(now <= expiry, "ZOM::delegateBySig: signature expired");
1224        return _delegate(signatory, delegatee);
```

- Here you are checking that a blockNumber is less than number of block.

```
1247        */
1248        function getPriorVotes(address account, uint blockNumber)
1249            external
1250            view
1251            returns (uint256)
1252 ▾      {
1253            require(blockNumber < block.number, "ZOM::getPriorVotes:
1254
```

- Here you are checking that a msg.sender is devaddr.

```
1596        // update dev address by the previous dev
1597 ▾      function dev(address _devaddr) public {
1598            require(msg.sender == devaddr, "dev: wut?");
1599            devaddr = _devaddr;
1600        }
```

- ## Critical vulnerabilities found in the contract

=> No Critical vulnerabilities found

- ## Medium vulnerabilities found in the contract

=> No Medium vulnerabilities found

- ## Low severity vulnerabilities found
  - ### FileName:- ZOMHEALTH.sol

  - #### 7.1: Short address attack:-
  => This is not a big issue in solidity, because now a days is increased
  In the new solidity version. But it is good practice to
  Check   for   the   short   address.
  => After updating the version of solidity it's not mandatory.
  => In some functions you are not checking the value of the
  address parameter

    - Function:- isContract ('account')

```
102        */
103 ▾    function isContract(address account) internal view returns
104        // According to EIP-1052, 0x0 is the value returned for
105        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca8227
106        // for accounts without code, i.e. `keccak256('')`
107        bytes32 codehash;
108        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c
109        // solhint-disable-next-line no-inline-assembly
```

      - It's necessary to check the address value of "account". Because here you are passing whatever variable comes in "account" address from outside.

Function: - transfer ('recipient')

```
683
684 ▾     function transfer(address recipient, uint256 amount) public
685           // reduce 5% from the amount
686           uint256 amounttoburn = _getamount(amount);
687           // send rest amount to receiver
688           uint256 amounttosend = amount.sub(amounttoburn);
689
```

- It's necessary to check the address value of "recipient". Because here you are passing whatever variable comes in "recipient" address from outside.

Function: - transferFrom('sender', 'recipient')

```
710           - burn 5% and send rest
711       */
712 ▾     function transferFrom(address sender, address recipient,
713           // reduce 5% from the amount
714           uint256 amounttoburn = _getamount(amount);
715           // send rest amount to receiver
716           uint256 amounttosend = amount.sub(amounttoburn);
717
```

- It's necessary to check the addresses value of "sender", and "recipient". Because here you are passing whatever variables come in "sender", and "recipient" addresses from outside.

Function: - getPriorVotes ('account')

```
831       @return The number of votes the account had as of the giv
832       */
833       function getPriorVotes(address account, uint blockNumber)
834           external
835           view
836           returns (uint256)
837 ▾     {
838           require(blockNumber < block.number, "ZOM::getPriorVotes:
839
```
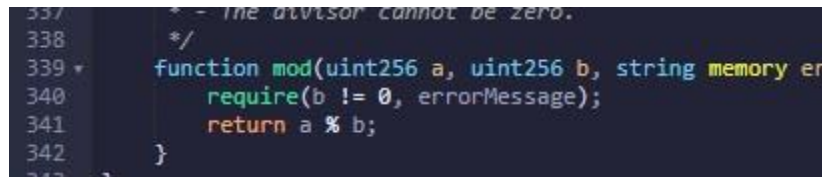
- It's necessary to check the address value of "account". Because here you are passing whatever variable comes in "account" address from outside.

- ○ **7.2: Some method is not used from safeMath library (For all files):-**

  => You have implemented safeMath library in a smart contract.

  => I found that one method (mod) is unused from safeMath library.

  => You can remove that method to reduce your code.

```
337          * - The divisor cannot be zero.
338          */
339 ▾   function mod(uint256 a, uint256 b, string memory er
340            require(b != 0, errorMessage);
341            return a % b;
342       }
343    }
```

- ○ **7.3: Compiler version not fixed (For all files):-**

  => 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 0.5.0; //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.

- • **FileName:- ZOM_OBSIDIAN.sol**

- ○ **7.4: Short address attack:-**

  ▪

  => This is not a big issue in solidity, because now a days is increased
  In the new solidity version. But it is good practice to
  Check   for   the   short   address.

  => After updating the version of solidity it's not mandatory.

  => In some functions you are not checking the value of
  Address parameter

Function: - isContract ('account')

```
348 ▾    function isContract(address account) internal view returns
349         // This method relies in extcodesize, which returns 0
350         // construction, since the code is only stored at the
351         // constructor execution.
352
353         uint256 size;
354         // solhint-disable-next-line no-inline-assembly
```

- o It's necessary to check the address value of "account". Because here you are passing whatever variable comes in "account" address from outside.

Function: - sendValue ('recipient')

```
374       */
375 ▾    function sendValue(address payable recipient, uint256 amount
376         require(address(this).balance >= amount, "Address: insuf
377
378         // solhint-disable-next-line avoid-low-level-calls, avoi
379         (bool success, ) = recipient.call{ value: amount }("");
380         require(success, "Address: unable to send value, recipie
```

- o It's necessary to check the address value of "recipient". Because here you are passing whatever variable comes in "recipient" address from outside.

Function: - safeApprove ('spender')

```
718       */
719 ▾    function safeApprove(IERC20 token, address spender, uint256 value
720         // safeApprove should only be called when setting an initial
721         // or when resetting it to zero. To increase and decrease it,
722         // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
723         // solhint-disable-next-line max-line-length
724         require((value == 0) || (token.allowance(address(this), spend
```

- o It's necessary to check the address value of "spender". Because here you are passing whatever variable comes in "spender" address from outside.

Function: - safeIncreaseAllowance('spender')

```
729
730 ▾    function safeIncreaseAllowance(IERC20 token, address spender
731         uint256 newAllowance = token.allowance(address(this), s
732         _callOptionalReturn(token, abi.encodeWithSelector(token
733     }
734
```

- o It's necessary to check the address value of "spender". Because here you are passing whatever variable comes in "spender" address from outside.

Function: - safeDecreaseAllowance('spender')

```
734
735 ▾        function safeDecreaseAllowance(IERC20 token, address spender,
736              uint256 newAllowance = token.allowance(address(this), spe
737              _callOptionalReturn(token, abi.encodeWithSelector(token.a
738        }
739
```

- It's necessary to check the address value of "spender". Because here you are passing whatever variable comes in "spender" address from outside.

Function: - transferFrom('spender', 'recipient')

```
1126 ▾        function transferFrom(address sender, address recipient, uint25
1127              // reduce 5% from the amount
1128              uint256 amounttoburn = _getamount(amount);
1129              // send rest amount to receiver
1130              uint256 amounttosend = amount.sub(amounttoburn);
1131
1132              // burn 5%
```

- It's necessary to check the addresses value of "spender", "recipient". Because here you are passing whatever variable comes in "spender", "recipient" addresses from outside.

o 7.5: Unchecked return value or response:-

=> 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: - safedefitokentestTransferRA

```
1587 ▾        function safeZomTransfer(address _to, uint256 _amount)
1588              uint256 zomBal = zom.balanceOf(address(this));
1589 ▾          if (_amount > zomBal) {
1590                  zom.transfer(_to, zomBal);
1591 ▾          } else {
1592                  zom.transfer(_to, _amount);
1593              }
1594
```

- Here you are calling transfer method 2 times. It is good to check that the transfer is successfully done or not.

⬇ Function: - safeTransfer

```
702     using Address for address;
703
704 ▾   function safeTransfer(IERC20 token, address to, uint256 va
705         _callOptionalReturn(token, abi.encodeWithSelector(toke
706     }
707
```

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

⬇ Function: - safeTransferFrom

```
707
708 ▾   function safeTransferFrom(IERC20 token, address from, add
709         _callOptionalReturn(token, abi.encodeWithSelector(to
710     }
711
```

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

• Summary of the Audit

Overall the code is well and performs well.

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 ;)).

- • **Note:** Please focus on version of solidity (Use latest),
  check addresses, and check return value of transfer method.

- • I have seen that a developer is using block's timestamp and
  now method so, I like to tell you that write smart contracts
  with the notion that block values are not precise, and the
  use of them can lead to unexpected effects.