SMART CONTRACT AUDIT REPORT

for

SHUTTLEONE

Prepared By: Shuxiao Wang

Hangzhou, China

Apr. 6, 2020

Document Properties

Client	ShuttleOne
Title	Smart Contract Audit Report
Target	SZO Token Smart Contract
Version	0.5
Author	Chiachih Wu
Auditors	Huaguo Shi, Chiachih Wu
Reviewed by	Chiachih Wu
Approved by	Jeff Liu
Classification	Confidential

Version Info

Version	Date	Author	Description
0.5	Apr. 6, 2020	Chiachih Wu	More Findings Added
0.4	Mar. 12, 2020	Chiachih Wu	More Findings Added
0.3	Mar. 8, 2020	Chiachih Wu	More Findings Added
0.2	Feb. 10, 2020	Huaguo Shi	Status Update
0.1	Feb. 7, 2020	Huaguo Shi	Initial Draft

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Shuxiao Wang
Phone	+86 173 6454 5338
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the **SZO Token Smart Contract** design document and related smart contract source code, we in the report outline our systematic method to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistency between smart contract code and the white paper, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to ERC20-compliance, security, or performance. This document outlines our audit results.

1.1 About SZO Token Smart Contract

The basic information of SZO Token Smart Contract is as follows:

Table 1.1: Basic Information of SZO Token Smart Contract

Item	Description
Issuer	ShuttleOne
Token Name	ShuttleOne
Token Symbol	SZO
Decimals	18
Total Supply of Tokens	230,000,000 + 11,500,000 Issuance Per Year
Token Type	ERC20
Platform	Solidity
Audit Method	Whitebox
Audit Completion Date	Apr. 6, 2020

In the following, we show the list of reviewed contracts used in this audit:

- https://github.com/shuttle-one/moneyprotocol.git (81b3ca0)
- https://github.com/shuttle-one/moneyprotocol.git (c1fc855)
- https://github.com/shuttle-one/moneyprotocol.git (1f419a9)

- https://github.com/shuttle-one/moneyprotocol.git (ae80675)
- https://github.com/shuttle-one/moneyprotocol.git (89675e3)
- https://github.com/shuttle-one/moneyprotocol.git (076aba4)

1.2 About PeckShield

PeckShield Inc. [9] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystem by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [3]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk;

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

We perform the audit according to the following procedures:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Table 1.2: Vulnerability Severity Classification

		Likelihood		
		High	*Medium*	*Low*
Impact	*High*	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

1.4 Disclaimer

Note that this audit does not give any warranties on finding all possible security issues of the given smart contract(s), i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as an investment advice.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
	Approve / TransferFrom Race Condition
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

2 Findings

2.1 Summary

Severity	# of Findings	
Critical	0	
High	0	
Medium	4	■ ■ ■ ■
Low	4	■ ■ ■ ■
Informational	7	■ ■ ■ ■ ■ ■ ■
Total	15	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 4 medium-severity vulnerability, 4 low-severity vulnerabilities, and 7 informational recommendations.

Table 2.1: Key Audit Findings

ID	Severity	Title	Type	Status
PVE-001	Low	approve()/transferFrom() Race Condition	Vulnerability	Confirmed
PVE-002	Info	Missing Exception Handling	Recommendation	Fixed
PVE-003	Low	Rounding Error in Withdraw	Vulnerability	Fixed
PVE-004	Medium	Business Logic Error in Contract Constructor	Vulnerability	Fixed
PVE-005	Medium	Manipulatable Prices and Unfair Arbitrage	Vulnerability	
PVE-006	Info	Improper Event Emission	Recommendation	Fixed
PVE-007	Info	Optimization Suggestions	Recommendation	Fixed
PVE-008	Low	Contract Address Checking Error in transferOwnership	Vulnerability	Fixed
PVE-009	Info	Redundant transferFrom Logic in intTransfer	Recommendation	Confirmed
PVE-010	Medium	Business Logic Error in mintToken()	Vulnerability	Fixed
PVE-011	Info	Excessive Checks in resetMintCount()	Recommendation	Fixed
PVE-012	Medium	Business Logic Error in getKYCData()	Vulnerability	Fixed
PVE-013	Low	Missing Blacklist Checks in transferFrom()	Vulnerability	Fixed
PVE-014	Info	Redundant Getter Functions	Recommendation	Fixed
PVE-015	Info	owner v.s. owners	Recommendation	

Please refer to Chapter 3 for details.

3 Detailed Results

3.1 approve()/transferFrom() Race Condition

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Medium

Description

There is a known race condition issue regarding `approve()` / `transferFrom()` [2]. Specifically, when a user intends to reduce the allowed spending amount previously approved from, say, 10 SZO to 1 SZO. The previously approved spender might race to transfer the amount you initially approved (the 10 SZO) and then additionally spend the new amount you just approved (1 SZO). This breaks the user's intention of restricting the spender to the new amount (1 SZO), **not** the sum of old amount and new amount (11 SZO).

```
function approve(address _spender, uint256 _value)
    public returns (bool){
    allowed[msg.sender][_spender] = _value;

    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

Listing 3.1: XSEToken.sol

Recommendation Add additional sanity checks in `approve()` besides the given workaround functions `increaseApproval()/decreaseApproval()`.

```
function approve(address _spender, uint256 _value)
    public returns (bool){
    require((_value == 0) || (allowed[msg.sender][_spender] == 0));

    allowed[msg.sender][_spender] = _value;

    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

Listing 3.2: Revised XSEToken.sol

3.2 Missing Exception Handling

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A

Description

Within this contract, a specific function, `setProfitAddr()`, the custody of ShuttleOne that would be used during extreme price volatility of eth or SZO prices and firstRedemption event.

As a common best practice, it is necessary to process invalid parameters, and it will be output to the log.

```
    function setProfitAddr(uint256 addrIdx) public onlyOwners{
        if(addrIdx == 1){
            require(msg.sender != profitAddr2);
            profitAddr1 = msg.sender;
        }
        if(addrIdx == 2){
            require(msg.sender != profitAddr1);
            profitAddr2 = msg.sender;
        }
     }
```

Listing 3.3: XSEToken.sol

Recommendation Add invalid parameters handling and describe the corresponding error message.

```
    function setProfitAddr(uint256 addrIdx) public onlyOwners{
        require(addrIdx == 1 ||  addrIdx == 2, "Invalid addrIdx");
        if(addrIdx == 1){
            require(msg.sender != profitAddr2, "Duplicate Profit Addr");
            profitAddr1 = msg.sender;
        }
        if(addrIdx == 2){
            require(msg.sender != profitAddr1,  "Duplicate Profit Addr");
            profitAddr2 = msg.sender;
        }
    }
```

Listing 3.4: Revised XSEToken.sol

3.3 Rounding Error in Withdraw

- ID: PVE-003
- Severity: Low
- Likelihood: Medium
- Impact: Low

Description

In the function `withDrawFunc()`, which is used for withdrawal. fund is divided into two addresses and the amount of withdrawal is divided equally. There is a problem here. If the parameter is odd, a small part of fund cannot be extracted.

```
    function withDrawFunc(uint256 _fund) public onlyOwners{
    require(address(this).balance >= _fund);
    require(tokenProfit >= _fund);
    require(profitAddr1 != address(0));
    require(profitAddr2 != address(0));

        profitAddr1.transfer(_fund / 2);
        profitAddr2.transfer(_fund / 2);

    tokenProfit -= _fund;
  }
```

Listing 3.5: XSEToken.sol

Recommendation The profitAddr1 is taken as 1/2, and the rest fund remove to profitAddr2 .

```
    function withDrawFunc(uint256 _fund) public onlyOwners{
    require(address(this).balance >= _fund);
    require(tokenProfit >= _fund);
    require(profitAddr1 != address(0));
    require(profitAddr2 != address(0));

        profitAddr1.transfer(_fund / 2);
        profitAddr2.transfer(fund - (_fund / 2));

    tokenProfit -= _fund;
```

```
434 }
```

Listing 3.6: Revised XSEToken.sol

3.4 Business Logic Error in Contract Constructor

- ID: PVE-004
- Severity: Medium
- Likelihood: High
- Impact: Low
- Target: XSEToken.sol, README.md

Description

The nextMintTime set in contract constructor is not compatible to the design (i.e., there is a 5% inflation per year after the issuance of all the tokens in the genesis round). Specifically, in line 266, the nextMintTime should be startTime + 365 days.

```
259 constructor() public {
260
261     balance[lockContract] = tokenLock;
262     haveKYC[lockContract] = true;
263     MAX_TOKEN_SELL = HARD_CAP - tokenLock;
264     emit Transfer(address(this),lockContract,tokenLock);
265     startTime = now;
266     nextMintTime = startTime + 356 days;
267 }
```

Listing 3.7: XSEToken.sol

Recommendation Set the nextMintTime as startTime + 365 days, which is compatible to the design.

```
259 constructor() public {
260
261     balance[lockContract] = tokenLock;
262     haveKYC[lockContract] = true;
263     MAX_TOKEN_SELL = HARD_CAP - tokenLock;
264     emit Transfer(address(this),lockContract,tokenLock);
265     startTime = now;
266     nextMintTime = startTime + 365 days;
267 }
```

Listing 3.8: XSEToken.sol

3.5 Manipulatable Prices and Unfair Arbitrage

- ID: PVE-005
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

Description

There is a known tradeTrap [5] issue regarding setTokenPrice(). The setTokenPrice() function, protected by the onlyOwner modifier, merely allows the owner to set the sell price of the tokens. Owner can be an arbitrager of these tokens if she wants. As an example, the owner can make profits by buying lower price tokens with setTokenPrice() and buyToken() and selling those tokens as market price in the exchange. It's unfair to exchange users or the public token market with the privileged function for manipulating the price of SZO tokens.

```
 //Change token sell price.
  function setTokenPrice(uint256 pricePerToken) public onlyOwners returns(bool){
    require(pricePerToken > 400000000000000);

    token_price = pricePerToken;
    return true;
  }
```

Listing 3.9: XSEToken.sol

```
  function buyToken() payable public returns(bool){
      uint256 amount = msg.value / token_price;
      tokenProfit += (token_price - tokenRedeem) * amount;

      amount  = amount * _1Token;
      require(totalSell + amount <= MAX_TOKEN_SELL);

      totalSell += amount;
      totalSupply_ += amount;
      balance[msg.sender] += amount;
      emit Transfer(address(this),msg.sender,amount);
      return true;
  }
```

Listing 3.10: XSEToken.sol

Moreover, the redeemFee() function allows privileged users to sell tokens to the smart contract with tokenRedeem price. If the price at the market is lower than tokenRedeem, they can purchase cheaper tokens from the market, sell them to the smart contract, and make profits.

```
function redeemFee(uint256 amount) public onlyOwners returns(bool){
    uint256  _fund;
    require(agents[msg.sender] == true,"SZO/ERROR-not-agent");

    _fund = (amount / _1Token) * tokenRedeem;
    require(balance[msg.sender] >= amount,"SZO/ERROR-insufficient-balance-agent");
    require(address(this).balance >= _fund,"SZO/ERROR-insufficient-balance-szo");

    balance[msg.sender] -= amount;
    totalSupply_ -= amount; // burn token
    emit Transfer(msg.sender,address(0),amount);

    msg.sender.transfer(_fund);

    return true;
}
```

Listing 3.11: XSEToken.sol

Recommendation Add a function to disable `buyToken()`, `mintToken()`, and `redeemFee()` before SZO is listed on any exchange.

3.6 Improper Event Emission

- ID: PVE-006
- Severity: Informational
- Likelihood: Medium
- Impact: None

Description

Events are inheritable members of contracts. When they are called, the arguments be stored in the transaction's log - a special data structure in the blockchain. In redeemFee() and burn(), after burned token operation, event Transfer() is called to record the log(line 389), but Transfer event transferring the token to itself.

```
380  //Redeem token that use for fee. after reedeem token will burn
381  function redeemFee(uint256 amount) public onlyOwners returns(bool){
382    uint256  _fund;
383    _fund = (amount / _1Token) * tokenRedeem;
384    require(balance[msg.sender] >= amount);
385    require(address(this).balance >= _fund);
386
387    balance[msg.sender] -= amount;
388    totalSupply_ -= amount; // burn token
```

```
    emit Transfer(msg.sender,address(this),amount);

    msg.sender.transfer(_fund);

    return true;
  }

  function burn(uint256 amount) public onlyOwners returns(bool){
    require(balance[msg.sender] >= amount);

    balance[msg.sender] -= amount;
    totalSupply_ -= amount; // burn token
    emit Transfer(msg.sender,address(this),amount);

    return true;
  }
```

Listing 3.12: XSEToken.sol

Recommendation When the token is burned, the transfer paramaters 'to' should be set to 0x0. Keep blockchain logs consistent with execution results.

```
  //Redeem token that use for fee. after reedeem token will burn
  function redeemFee(uint256 amount) public onlyOwners returns(bool){
    uint256 _fund;
    _fund = (amount / _1Token) * tokenRedeem;
    require(balance[msg.sender] >= amount);
    require(address(this).balance >= _fund);

    balance[msg.sender] -= amount;
    totalSupply_ -= amount; // burn token
    emit Transfer(msg.sender,address(0),amount);

    msg.sender.transfer(_fund);

    return true;
  }

  function burn(uint256 amount) public onlyOwners returns(bool){
    require(balance[msg.sender] >= amount);

    balance[msg.sender] -= amount;
    totalSupply_ -= amount; // burn token
    emit Transfer(msg.sender,address(0),amount);

    return true;
  }
```

Listing 3.13: Revised XSEToken.sol

3.7 Optimization Suggestions

- ID: PVE-007
- Severity: Informational
- Likelihood: N/A
- Impact: N/A

Description

In line 373 of setTokenPrice(), the constant value 400000000000000 is used to validate the input parameter, pricePerToken. However, according to the design, the constant value is identical to the variable tokenRedeem. In software convention, we don't define the same parameter in multiple places, which may cause maintenance issue in the future.

```
371  //Change token sell price.
372    function setTokenPrice(uint256 pricePerToken) public onlyOwners returns(bool){
373      require(pricePerToken > 400000000000000);
374
375      token_price = pricePerToken;
376      return true;
377    }
```

Listing 3.14: XSEToken.sol

Recommendation Use tokenRedeem instead of constant 400000000000000.

```
371  //Change token sell price.
372    function setTokenPrice(uint256 pricePerToken) public onlyOwners returns(bool){
373      require(pricePerToken > tokenRedeem);
374
375      token_price = pricePerToken;
376      return true;
377    }
```

Listing 3.15: Revised XSEToken.sol

Another optimization suggestion is in the buyToken() function. Any user can purchase SZO tokens through buyToken() with msg.value of eth. The amount of tokens being purchased is calculated in line 273. However, when amount == 0 (i.e., msg.value is not enough), the buyToken() function would perform zero SZO token issuance with a Transfer event emitted, which is a waste of gas without meaningful log generated. Furthermore, it's unfair to the buyers who pay not enough ether into buyToken() due to the fact that those msg.value ether would not be refunded and book-kept. The smart contract would keep those leftovers forever.

```
272    function buyToken() payable public returns(bool){
273        uint256 amount = msg.value / token_price;
274        tokenProfit += (token_price - tokenRedeem) * amount;
```

```

        amount  = amount * _1Token;
        require(totalSell + amount <= MAX_TOKEN_SELL);

        totalSell += amount;
        totalSupply_ += amount;
        balance[msg.sender] += amount;
        emit Transfer(address(this),msg.sender,amount);
        return true;
    }
```

Listing 3.16: XSEToken.sol

Recommendation Ensure `amount > 0` before the rest of logic in `buyToken()`.

```
    function buyToken() payable public returns(bool){
        uint256 amount = msg.value / token_price;
        require(amount > 0, "Insufficient eth");
        tokenProfit += (token_price - tokenRedeem) * amount;

        amount  = amount * _1Token;
        require(totalSell + amount <= MAX_TOKEN_SELL);

        totalSell += amount;
        totalSupply_ += amount;
        balance[msg.sender] += amount;
        emit Transfer(address(this),msg.sender,amount);
        return true;
    }
```

Listing 3.17: XSEToken.sol

3.8 Contract Address Checking Error in transferOwnership

- ID: PVE-008
- Severity: Low
- Likelihood: Low
- Impact: Medium

Description

In `transferOwnership()`, the `isContract()` function is used to check if `newOwner` is a contract address (line 80).

```
79    function transferOwnership(address  newOwner, string memory newOwnerName) public
          onlyOwner{
```

```
    require(isContract(newOwner) == false);
    uint256 idx;
    if(ownerToProfile[newOwner] == 0)
    {
        idx = ownerName.push(newOwnerName);
        ownerToProfile[newOwner] = idx;
    }


    emit OwnershipTransferred(owner,newOwner);
    owner = newOwner;

  }
```

Listing 3.18: XSEToken.sol

Inside `isContract()`, the code size of `_addr` is retrieved by `extcodesize` in line 58 of the following code snippets.

```
55    function isContract(address _addr) internal view returns(bool){
56       uint256 length;
57       assembly{
58        length := extcodesize(_addr)
59       }
60       if(length > 0){
61         return true;
62     }
63     else {
64       return false;
65     }
66
67    }
```

Listing 3.19: XSEToken.sol

However, due to the fact that the contract address created by an existing contract is predictable in Ethereum. If the `newOwner` is a pre-calculated but not yet created contract address, the `isContract()` function has no chance to detect that.

Recommendation A good practice of implementing ownership transfer is using the two-phase transfer mechanism. In the first phase, the candidate of the new owner is set. Later on, in the second phase, the candidate signs and sends out another transaction to claim the previous nomination. It would be a good chance to detect if the candidate address is a contract address or not.

A sample implementation is in the following:

```
  function transferOwnership(address newOwner, string memory newOwnerName) public
      onlyOwner{
    candidateOwner = newOwner;
    candidateOwnerName = newOwnerName;
  }

```

```
function claimOwnership() public {
  require(candidateOwner == msg.sender);
  require(isContract(candidateOwner) == false);
  uint256 idx;
  if(ownerToProfile[candidateOwner] == 0)
  {
      idx = ownerName.push(candidateOwnerName);
      ownerToProfile[candidateOwner] = idx;
  }


  emit OwnershipTransferred(owner,candidateOwner);
  owner = candidateOwner;
  candidateOwner = address(0);

}
```

Listing 3.20: XSEToken.sol

Also, the `isContract()` should be fixed as following to prevent the case that the candidate owner calls the `claimOwnership()` inside a contract constructor, which would trick the `extcodesize` to return 0 as the contract itself has not been contructed yet.

```
function isContract(address _addr) internal view returns(bool){
  if (tx.origin == msg.sender) {
    return false;
}
  else {
    return true;
  }
}
```

Listing 3.21: XSEToken.sol

3.9 Redundant transferFrom Logic in intTransfer

- ID: PVE-009
- Severity: Informational
- Likelihood: N/A
- Impact: N/A

Description

Both Method `intTransfer()` and `transferFrom()` of ERC20 implement transfer tokens. Erc20 has provided a complete transfer APIs, such as `transferFrom()` and `approve()`. It does not require additional development of transfer functions. so we consider that intTransfer () and transferFrom() of ERC20 have duplicate function.

```
//Add on KYC check to StandarERC20 transferFrom function
function transferFrom(address _from, address _to, uint256 _value) public returns (bool){

    require(haveKYC[_from] == true);
//        require(haveKYC[_to] == true); // remove recieve no KYC
    super.transferFrom(_from, _to, _value);
 }
  // Set address can allow internal transfer or not. Default are off. Owner of address should allow by them self
function setAllowInterTransfer(bool _allow) public returns(bool){
  haveInterTran[msg.sender] = _allow;
  return true;
}



// This function use only for internal wallet that create by XSE Wallet only
// sender and reciever  will need to KYC
 function intTransfer(address _from, address _to, uint256 _value) external onlyOwners returns(bool){

  require(haveInterTran[_from] == true);
  require(balance[_from] >= _value);
  require(_to != address(0));
  require(haveKYC[_from] == true);
//  require(haveKYC[_to] == true);

  balance[_from] -= _value;
  balance[_to] += _value;

  emit Transfer(_from,_to,_value);
  return true;
}
```

Listing 3.22: XSEToken.sol

Recommendation

We think there are two solutions:

1) Functoins `intTransfer()` and `setAllowInterTransfer()` do exist at all. You can do the same implement in `transferfrom()` / `approve()`;

2) Calling `approve()` in `setAllowInterTransfer()` and calling `super.transferFrom()` in `intTransfer ()`.

3.10 Business Logic Error in mintToken()

- ID: PVE-010
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

Description

In `mintToken()`, the global variable `mintCount` is used to check if the amount of minted tokens exceeds `MINT_PER_YEAR`. However, the implementation fails to update `mintCount` such that the limitation is not actually working. Also, any `msg.value` which is less than `token_price` is meaningless here.

```solidity
  function mintToken() public payable canMintToken returns(bool){
     require(haveKYC[msg.sender] == true);

     uint256 amount = msg.value / token_price;
     tokenProfit += (token_price - tokenRedeem) * amount;
     amount  = amount * _1Token;

     require(mintCount + amount <= MINT_PER_YEAR);
     totalSupply_ += amount;
     balance[msg.sender] += amount;
     emit Transfer(address(0),msg.sender,amount);
     return true;
  }
```

Listing 3.23: XSEToken.sol

Recommendation Update `mintCount` and validate `msg.value`.

```solidity
  function mintToken() public payable canMintToken returns(bool){
     require(haveKYC[msg.sender] == true);
     require(msg.value >= token_price);

     uint256 amount = msg.value / token_price;
     tokenProfit += (token_price - tokenRedeem) * amount;
     amount  = amount * _1Token;

     require(mintCount + amount <= MINT_PER_YEAR);
     mintCount += amount;
     totalSupply_ += amount;
     balance[msg.sender] += amount;
     emit Transfer(address(0),msg.sender,amount);
     return true;
  }
```

Listing 3.24: XSEToken.sol

3.11 Excessive Checks in resetMintCount()

- ID: PVE-011
- Severity: Informational
- Likelihood: N/A
- Impact: N/A

Description

In `resetMintCount()`, the variable `totalSell` is used to check if the amount of tokens sold in `buyToken()` reaches `MAX_TOKEN_SELL` (line 338).

```
337     function resetMintCount() public onlyOwners returns(bool) {
338         if(now > nextMintTime && MINT_PER_YEAR == mintCount && totalSell >=
                MAX_TOKEN_SELL){
339             nextMintTime = nextMintTime + 365;
340             mintCount = 0;
341            return true;
342         }
343 
344         return false;
345     }
```

Listing 3.25: XSEToken.sol

However, the implementation of `buyToken()` can only increment `totalSell` to `MAX_TOKEN_SELL` (line 294), which means the case `totalSell > MAX_TOKEN_SELL` is not possible.

```
285 function buyToken() payable public returns(bool){
286     require(msg.value >= token_price);
287     require(now - nextBuyTime > 60 seconds);
288 
289     uint256 amount = msg.value / token_price;
290     tokenProfit += (token_price - tokenRedeem) * amount;
291 
292 
293     amount  = amount * _1Token;
294     require(totalSell + amount <= MAX_TOKEN_SELL);
295 
296     totalSell += amount;
297     totalSupply_ += amount;
298     balance[msg.sender] += amount;
299     emit Transfer(address(this),msg.sender,amount);
300     return true;
301 }
```

Listing 3.26: XSEToken.sol

Recommendation Remove the `totalSell > MAX_TOKEN_SELL` check.

```
    function resetMintCount() public onlyOwners returns(bool) {
        if(now > nextMintTime && MINT_PER_YEAR == mintCount && totalSell == MAX_TOKEN_SELL){
            nextMintTime = nextMintTime + 365;
            mintCount = 0;
            return true;
        }

        return false;
    }
```

Listing 3.27: XSEToken.sol

3.12 Business Logic Error in getKYCData()

- ID: PVE-012
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

Description

In `createKYCData()`, the `kycDatas` array stores the newly created KYC with the `id` stored in `OwnerToKycData` . However, the `id` is actually the length of the `kycDatas` array after the `push` operation in line 420, which means the index of the newly created KYC is stored at `id-1` in `OwnerToKycData`.

```
417    function createKYCData(bytes32 _KycData1, bytes32 _kycData2, address  _wallet) onlyOwners public returns(uint256){
418        require(haveKYC[_wallet] == false); // can't re KYC  if already KYC
419
420        uint256 id = kycDatas.push(KYCData(_KycData1, _kycData2));
421        OwnerToKycData[_wallet] = id;
422        haveKYC[_wallet] = true;
423
424        return id;
425    }
```

Listing 3.28: XSEToken.sol

In `getKYCData()`, the `index` retrieved from `OwnerToKycData` is used to retrieve `_data1` and `_data2` in line 432 and 433, which fails to get the accurate KYC as mentioned above.

```
428    function getKYCData(address _wallet) public view returns(bytes32 _data1,bytes32 _data2){
        require(haveKYC[_wallet] == true);
        uint256 index = OwnerToKycData[_wallet];

        _data1 = kycDatas[index].KYCData01;
        _data2 = kycDatas[index].KYCData02;
    }
```

Listing 3.29: XSEToken.sol

Recommendation Fix the `index` in `getKYCData()`.

```
    function getKYCData(address _wallet) public view returns(bytes32 _data1,bytes32
        _data2){
        require(haveKYC[_wallet] == true);
        uint256 index = OwnerToKycData[_wallet] - 1;

        _data1 = kycDatas[index].KYCData01;
        _data2 = kycDatas[index].KYCData02;
    }
```

Listing 3.30: XSEToken.sol

3.13 Missing Blacklist Checks in transferFrom()

- ID: PVE-013
- Severity: Low
- Likelihood: Low
- Impact: Medium

Description

In `transferFrom()`, the `msg.sender` and `_to` are checked to ensure that they are not in the blacklist. However, the check against the `_from` address is missed here, which results in blacklisted accounts transferring tokens out.

```
  function transferFrom(address _from, address _to, uint256 _value) public returns (bool
      ){

        require(haveKYC[_from] == true);
        require(blacklist[msg.sender] == false);
        require(blacklist[_to] == false);

//      require(haveKYC[_to] == true); // remove recieve no KYC
        super.transferFrom(_from, _to, _value);
  }
```

Listing 3.31: XSEToken.sol

Recommendation Add blacklist check against `_from`.

```
  function transferFrom(address _from, address _to, uint256 _value) public returns (bool){

        require(haveKYC[_from] == true);
        require(blacklist[msg.sender] == false);
        require(blacklist[_from] == false);
        require(blacklist[_to] == false);

//      require(haveKYC[_to] == true); // remove recieve no KYC
        super.transferFrom(_from, _to, _value);
  }
```

Listing 3.32: XSEToken.sol

3.14 Redundant Getter Functions

- ID: PVE-014
- Severity: Informational
- Likelihood: N/A
- Impact: N/A

Description

The following `view` functions are redundant due to the fact that `public` variables have built-in getter functions.

```
    function haveWhiteList(address _walletAddress) public view returns (bool){
        return whitelist[_walletAddress];
    }

    function haveBlackList(address _walletAddress) public view returns (bool){
        return blacklist[_walletAddress];
    }

    function haveShuttleOneWallet(address _walletAddress) public view returns (bool){
        return shuttleOneWallets[_walletAddress];
    }
```

Listing 3.33: XSEToken.sol

Recommendation Remove the redundant getter functions and make `whitelist`, `blacklist`, and `shuttleOneWallets` public variables.

3.15 owner v.s. owners

- ID: PVE-015
- Severity: Informational
- Likelihood: N/A
- Impact: N/A

Description

Throughout the SZO Token Smart Contract, the only place that check the permission of `owner` is `transferOwnership()` which is used to transfer the ownership to a new address. However, there are quite a few places which validate if the `msg.sender` is in `owners` using the `onlyOwners()` modifier. Here, the existence of `owner` is not clear but at least it should be one of the `owners`.

```
modifier onlyOwners(){
  require(owners[msg.sender] == true);
  _;
}
```

Listing 3.34: XSEToken.sol

Recommendation Allow `owner` to pass the `onlyOwners` check. On the other hand, we suggest to remove the `owner` if it is not necessary.

```
modifier onlyOwners(){
  require(msg.sender == owner || owners[msg.sender] == true);
  _;
}
```

Listing 3.35: XSEToken.sol

3.16 Other Suggestions

Due to the fact that compiler upgrades might bring unexpected compatibility or inter-version consistencies, it is always suggested to use fixed compiler versions whenever possible. As an example, we highly encourage to explicitly indicate the Solidity compiler version, e.g., `pragma solidity 0.5.10;` instead of `pragma solidity ^0.5.10;`.

Moreover, we strongly suggest not to use experimental Solidity features or third-party unaudited libraries. If necessary, refactor current code base to only use stable features or trusted libraries. In case there is an absolute need of leveraging experimental features or integrating external libraries, make necessary contingency plans.

4 Conclusion

The SZO Token Smart Contract was analyzed in this audit. No critical or high level vulnerabilities had been discovered so far, though there exists quite a few medium or low issues that could be exploited in the wild. Meanwhile, as disclaimed in Section 1.4, we appreciate any constructive feedbacks or suggestions.

5 | Appendix

5.1 Basic Coding Bugs

5.1.1 Constructor Mismatch

- Description: Whether the contract name and its constructor are not identical to each other.
- Result: Not found
- Severity: Critical

5.1.2 Ownership Takeover

- Description: Whether the set owner function is not protected.
- Result: Not found
- Severity: Critical

5.1.3 Redundant Fallback Function

- Description: Whether the contract has a redundant fallback function.
- Result: Not found
- Severity: Critical

5.1.4 Overflows & Underflows

- Description: Whether the contract has general overflow or underflow vulnerabilities [4, 6, 7, 8, 10].
- Result: Not found
- Severity: Critical

5.1.5 Reentrancy

- Description: Reentrancy [11] is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
- Result: Not found
- Severity: Critical

5.1.6 Money-Giving Bug

- Description: Whether the contract returns funds to an arbitrary address.
- Result: Not found
- Severity: High

5.1.7 Blackhole

- Description: Whether the contract locks ETH indefinitely: merely in without out.
- Result: Not found
- Severity: High

5.1.8 Unauthorized Self-Destruct

- Description: Whether the contract can be killed by any arbitrary address.
- Result: Not found
- Severity: Medium

5.1.9 Revert DoS

- Description: Whether the contract is vulnerable to DoS attack because of unexpected `revert`.
- Result: Not found
- Severity: Medium

5.1.10 Unchecked External Call

- Description: Whether the contract has any external call without checking the return value.
- Result: Not found
- Severity: Medium

5.1.11 Gasless Send

- Description: Whether the contract is vulnerable to gasless send.
- Result: Not found
- Severity: Medium

5.1.12 Send Instead of Transfer

- Description: Whether the contract uses send instead of transfer.
- Result: Not found
- Severity: Medium

5.1.13 Costly Loop

- Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
- Result: Not found
- Severity: Medium

5.1.14 (Unsafe) Use of Untrusted Libraries

- Description: Whether the contract use any suspicious libraries.
- Result: Not found
- Severity: Medium

5.1.15 (Unsafe) Use of Predictable Variables

- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: Not found
- Severity: Medium

5.1.16 Transaction Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Not found
- Severity: Medium

5.1.17 Deprecated Uses

- Description: Whether the contract use the deprecated `tx.origin` to perform the authorization.
- Result: Not found
- Severity: Medium

5.2 Semantic Consistency Checks

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: Not found
- Severity: Critical

5.3 Additional Recommendations

5.3.1 Avoid Use of Variadic Byte Array

- Description: Use fixed-size byte array is better than that of `byte[]`, as the latter is a waste of space.
- Result: Not found
- Severity: Low

5.3.2 Make Visibility Level Explicit

- Description: Assign explicit visibility specifiers for functions and state variables.
- Result: Not found
- Severity: Low

5.3.3 Make Type Inference Explicit

- Description: Do not use keyword `var` to specify the type, i.e., it asks the compiler to deduce the type, which is not safe especially in a loop.
- Result: Not found
- Severity: Low

5.3.4 Adhere To Function Declaration Strictly

- Description: Solidity compiler (version 0.4.23) enforces strict ABI length checks for return data from `calls()` [1], which may break the the execution if the function implementation does NOT follow its declaration (e.g., no return in implementing `transfer()` of ERC20 tokens).
- Result: Not found
- Severity: Low

References

[1] axic. Enforcing ABI length checks for return data from calls can be breaking. https://github.com/ethereum/solidity/issues/4116.

[2] HaleTom. Resolution on the EIP20 API Approve / TransferFrom multiple withdrawal attack. https://github.com/ethereum/EIPs/issues/738.

[3] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[4] PeckShield. ALERT: New batchOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018-10299). https://www.peckshield.com/2018/04/22/batchOverflow/.

[5] PeckShield. Full Disclosure of Highly-Manipulatable, tradeTrap-Affected ERC20 Tokens in Multiple Top Exchanges . https://blog.peckshield.com/2018/06/11/tradeTrap/.

[6] PeckShield. New burnOverflow Bug Identified in Multiple ERC20 Smart Contracts (CVE-2018-11239). https://www.peckshield.com/2018/05/18/burnOverflow/.

[7] PeckShield. New multiOverflow Bug Identified in Multiple ERC20 Smart Contracts (CVE-2018-10706). https://www.peckshield.com/2018/05/10/multiOverflow/.

[8] PeckShield. New proxyOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018-10376). https://www.peckshield.com/2018/04/25/proxyOverflow/.

[9] PeckShield. PeckShield Inc. https://www.peckshield.com.

[10] PeckShield. Your Tokens Are Mine: A Suspicious Scam Token in A Top Exchange. https://www.peckshield.com/2018/04/28/transferFlaw/.

[11] Solidity. Warnings of Expressions and Control Structures. http://solidity.readthedocs.io/en/develop/control-structures.html.