SMART CONTRACT AUDIT REPORT

for

DFORCE NETWORK

Prepared By: Shuxiao Wang

Hangzhou, China
Mar. 30, 2020

Document Properties

Client	dForce Network
Title	Smart Contract Audit Report
Target	X-Swap
Version	1.0-rc1
Author	Chiachih Wu
Auditors	Chiachih Wu, Huaguo Shi
Reviewed by	Chiachih Wu
Approved by	Xuxian Jiang
Classification	Confidential

Version Info

Version	Date	Author(s)	Description
1.0-rc1	Mar. 30, 2020	Chiachih Wu	Status Update
0.1	Mar. 28, 2020	Chiachih Wu	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 5
1.1 About X-Swap . . . 5
1.2 About PeckShield . . . 6
1.3 Methodology . . . 6
1.4 Disclaimer . . . 8

2 Findings 10
2.1 Summary . . . 10
2.2 Key Findings . . . 11

3 Detailed Results 12
3.1 Potential Integer Overflow in trade() . . . 12
3.2 Missing Same Token Swapping Checks . . . 13
3.3 Missing Owner Check in transferOwnership() . . . 14
3.4 Missing Assets Migration in setLendFMe() . . . 15
3.5 Insufficient Approval in enableLending() . . . 15
3.6 Excessive Fallback Logic . . . 17
3.7 Business Logic Issue in trade() . . . 18
3.8 Other Suggestions . . . 19

4 Conclusion 20

5 Appendix 21
5.1 Basic Coding Bugs . . . 21
5.1.1 Constructor Mismatch . . . 21
5.1.2 Ownership Takeover . . . 21
5.1.3 Redundant Fallback Function . . . 21
5.1.4 Overflows & Underflows . . . 21
5.1.5 Reentrancy . . . 22
5.1.6 Money-Giving Bug . . . 22

5.1.7 Blackhole . . . 22
5.1.8 Unauthorized Self-Destruct . . . 22
5.1.9 Revert DoS . . . 22
5.1.10 Unchecked External `Call` . . . 23
5.1.11 Gasless `Send` . . . 23
5.1.12 `Send` Instead Of `Transfer` . . . 23
5.1.13 Costly Loop . . . 23
5.1.14 (Unsafe) Use Of Untrusted Libraries . . . 23
5.1.15 (Unsafe) Use Of Predictable Variables . . . 24
5.1.16 Transaction Ordering Dependence . . . 24
5.1.17 Deprecated Uses . . . 24
5.2 Semantic Consistency Checks . . . 24
5.3 Additional Recommendations . . . 24
5.3.1 Avoid Use of Variadic Byte Array . . . 24
5.3.2 Make Visibility Level Explicit . . . 25
5.3.3 Make Type Inference Explicit . . . 25
5.3.4 Adhere To Function Declaration Strictly . . . 25

References **26**

1 Introduction

Given the opportunity to review the **X-Swap** design document and related smart contract source code, we in the report outline our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, 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 either security or performance. This document outlines our audit results.

1.1 About X-Swap

X-Swap is designed to facilitate bi-directional swaps between stablecoins including USDx, USDT, USDC, TUSD, DAI, PAX, etc. The administrator maintains the exchange rate with a small amount of fee charged to each transaction. Stablecoin reserves will be supplied to Lendf.Me (lending protocol of dForce Network) for interest-generation.

The basic information of X-Swap is as follows:

Table 1.1: Basic Information of X-Swap

Item	Description
Issuer	dForce Network
Website	https://dforce.network/
Type	Ethereum Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	Mar. 30, 2020

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit:

- https://github.com/dforce-network/xswap/tree/audit (326a324)
- https://github.com/dforce-network/xswap/tree/audit (e0aaa54)

1.2 About PeckShield

PeckShield Inc. [18] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems 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).

Table 1.2: Vulnerability Severity Classification

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

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [13]:

- 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.

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 or analysis 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

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
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

additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- 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.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [12], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings.

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 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 Findings

2.1 Summary

Here is a summary of our findings after analyzing the X-Swap implementation. During the first phase of our audit, we studied the smart contract source code and ran our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	0	
Low	2	■■
Informational	5	■■■■■
Total	7	

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 2 low-severity vulnerabilities, and 5 informational recommendations.

Table 2.1: Key Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Info.	Potential Integer Overflow in trade()	Numeric Errors	Confirmed
PVE-002	Info.	Missing Same Token Swapping Checks	Error Conditions	Confirmed
PVE-003	Info.	Missing Owner Checks in transferOwnership()	Error Conditions	Confirmed
PVE-004	Low	Missing Assets Migration in setLendFMe()	Behavioral Problems	Confirmed
PVE-005	Low	Insufficient Approval in enableLending()	Privilege Issues	Resolved
PVE-006	Info.	Excessive Fallback Logic	Coding Practices	Confirmed
PVE-007	Info.	Business Logic Issue in trade()	Behavioral Problems	Confirmed

Please refer to Section 3 for details.

3 | Detailed Results

3.1 Potential Integer Overflow in trade()

- ID: PVE-001
- Severity: Informational
- Likelihood: Low
- Impact: None
- Target: `contracts/XSwap.sol`
- Category: Numeric Errors [10]
- CWE subcategory: CWE-190 [2]

Description

In `trade()`, the `_inputAmount` is scaled up to a number with 18 decimal places before calculating the `_amountToUser`. Within the calculation, the amount of `_amountToUser` less than 1 `_output` token is removed. This causes a precision problem that more than enough `_input` tokens are taken to swap for the `_output` tokens. For example, the price for swapping USDx for DAI is 0.97 and the caller pays in 10,000 USDx (i.e., $\_inputAmount = 10 \times 10^{21}$, $prices = 0.97 \times 10^{18}$). The `_tokenAmount` in line 75 would be $(10{\times}10^{21}){\times}(0.97{\times}10^{18}){\div}10^{18} = 9700{\times}10^{18}$. Since the `setPrices()` function allows privileged users to set the number (e.g., $0.97{\times}10^{18}$) to an arbitrary value, the calculation in line 75 may overflow in extreme cases. An extremem example could be exchaning 10 million BTC to corresponding ZWD (the Zimbabwean dollar introduced in 1980). The math would be $10^7 BTC \sim 10k \times 10^7 USD = 10^{11} USD \sim 300k \times 10^{11} ZWL = 3 \times 10^{16} ZWL \sim 3 \times 10^{16} \times 10^{25} ZWD = 3 \times 10^{41} ZWD$. Since both BTC and ZWD tokens would be scaled up to 18 decimal places, the multiplication in line 75 would be $3 \times 10^{41} \times 10^{36} = 3 \times 10^{77} > 2^{256}$.

```
60     function trade(address _input, address _output, uint256 _inputAmount, address
           _receiver) public returns (bool) {
61         require(isOpen, "not open");
62         require(prices[_input][_output] != 0, "invalid token address");
63         require(decimals[_input] != 0, "input decimal not setteled");
64         require(decimals[_output] != 0, "output decimal not setteled");
65
66         NonStandardIERC20Token(_input).transferFrom(msg.sender, address(this),
               _inputAmount);
```

```
        if(supportLending[_input]) {
            if (_input == dai) {
                IChai(chai).join(address(this), _inputAmount);
                ILendFMe(lendFMe).supply(chai, IERC20Token(chai).balanceOf(address(this)
                    ));
            }
            else
                ILendFMe(lendFMe).supply(_input, _inputAmount);
        }
        uint256 _tokenAmount = normalizeToken(_input, _inputAmount).mul(prices[_input][
            _output]).div(OFFSET);
```

Listing 3.1: contracts/XSwap.sol

```
    function setPrices(address _input, address _output, uint256 _price) public auth
        returns (bool) {
        prices[_input][_output] = _price;
        return true;
    }
```

Listing 3.2: contracts/XSwap.sol

Recommendation Validate the price with a reasonable range and decimals of `_input` and `_output`.

3.2 Missing Same Token Swapping Checks

- ID: PVE-002
- Severity: Informational
- Likelihood: None
- Impact: None
- Target: `contracts/USR.sol`
- Category: Error Conditions [9]
- CWE subcategory: CWE-248 [3]

Description

If the admin accidently sets the price of the same token, the `trade()` function would allow users to swap between two identical tokens. Specifically, `prices[_input][_output] != 0` is checkd in line 62 to prevent the caller from trading between non-supported tokens.

```
60     function trade(address _input, address _output, uint256 _inputAmount, address
           _receiver) public returns (bool) {
61         require(isOpen, "not open");
62         require(prices[_input][_output] != 0, "invalid token address");
63         require(decimals[_input] != 0, "input decimal not setteled");
64         require(decimals[_output] != 0, "output decimal not setteled");
```

Listing 3.3: contracts/XSwap.sol

Recommendation Ensure `_input != _output` in `setPrices()`. Furthermore, the same `_price` could be avoided by checking `prices[_input][_output] != _price` to save some gas.

```
function setPrices(address _input, address _output, uint256 _price) public auth
    returns (bool) {
    require(_input != _output, "invalid token address");
    require(_price != prices[_input][_output], "setting same price");
    prices[_input][_output] = _price;
    return true;
}
```

Listing 3.4: contracts/XSwap.sol

3.3 Missing Owner Check in transferOwnership()

- ID: PVE-003
- Severity: Informational
- Likelihood: None
- Impact: None
- Target: `contracts/DSLibrary/DSAuth.sol`
- Category: Error Conditions [9]
- CWE subcategory: CWE-248 [3]

Description

In the `DSAuth` contract, the `transferOwnership()` function does not validate the `newOwner_` against the `newOwner`, which is a waste of gas.

```
function transferOwnership(address newOwner_) public onlyOwner {
    require(newOwner_ != owner, "TransferOwnership: the same owner.");
    newOwner = newOwner_;
}
```

Listing 3.5: contracts/DSLibrary/DSAuth.sol

Recommendation Ensure `newOwner_ != newOwner`.

```
function transferOwnership(address newOwner_) public onlyOwner {
    require(newOwner_ != owner, "TransferOwnership: the same owner.");
    require(newOwner_ != newOwner, "TransferOwnership: the same newOwner.");
    newOwner = newOwner_;
}
```

Listing 3.6: contracts/DSLibrary/DSAuth.sol

3.4 Missing Assets Migration in setLendFMe()

- ID: PVE-004
- Severity: Low
- Likelihood: Low
- Impact: Medium
- Target: `contracts/XSwap.sol`
- Category: Behavioral Problems [8]
- CWE subcategory: CWE-841 [6]

Description

The `setLendFMe()` function allows the authenticated user to change the `lendFMe` to a new address. However, the old address may have some states and digital assets belongs to the X-Swap contract which should be migrated to the new `lendFMe` before setting the global address to the new one.

```
    function setLendFMe(address _lendFMe) public auth returns (bool) {
        lendFMe = _lendFMe;
        return true;
    }
```

Listing 3.7: contracts/XSwap.sol

Recommendation Keep an array of tokens which have been added into `supportLending[]`. For each token in the array, perform `disableLending()` to withdraw the digital assets back to X-Swap. After the assets migration is done, `setLendFMe()` can be called to change the `lendFMe` address. Then, we need to `enableLending()` all tokens in the previous array to complete the states migration. In the meantime, the `isOpen` flag should be turned off to avoid users from trading with X-Swap with the intermidiate states.

```
    function setLendFMe(address _lendFMe) public auth returns (bool) {
        require(isOpen != true, "should stop trading before setting new lendFMe");
        lendFMe = _lendFMe;
        return true;
    }
```

Listing 3.8: contracts/XSwap.sol

3.5 Insufficient Approval in enableLending()

- ID: PVE-005
- Severity: Low
- Likelihood: Low
- Impact: Medium
- Target: `contracts/XSwap.sol`
- Category: Privilege Issues [11]
- CWE subcategory: CWE-280 [4]

Description

In `enableLending()`, an ERC20 `_token` is put in the list of assets for lending such that all the following `trade()` and `transferIn()` calls with `_token` will be supplied into LendFMe. To achieve that, the allowance for `lendFMe` is set to the maximum value (i.e., `uint256(-1)`). However, in line 110, `allowance == 0` is checked before setting the maximum allowance. If, by any chance, the allowance is not set to `uint256(-1)`, the following `trade()` calls may revert while supplying `_token` into LendFMe.

```
101     function enableLending(address _token) public auth returns (bool) {
102         require(!supportLending[_token], "the token is already supported lending");
103         supportLending[_token] = true;
104 
105         if (_token == dai) {
106             IERC20Token(_token).approve(chai, uint256(-1));
107             IERC20Token(chai).approve(lendFMe, uint256(-1));
108         }
109         else {
110             if (NonStandardIERC20Token(_token).allowance(address(this), lendFMe) == 0)
111                 NonStandardIERC20Token(_token).approve(lendFMe, uint256(-1));
112         }
113 
114         uint256 _balance = NonStandardIERC20Token(_token).balanceOf(address(this));
115         if(_balance > 0) {
116             if (_token == dai) {
117                 IChai(chai).join(address(this), _balance);
118                 ILendFMe(lendFMe).supply(chai, IERC20Token(chai).balanceOf(address(this)));
119             }
120             else
121                 ILendFMe(lendFMe).supply(_token, _balance);
122         }
123         return true;
124     }
```

Listing 3.9: contracts/XSwap.sol

Recommendation Remove the `allowance == 0` check. Note that HBTC has upgraded the `HBTCStorage` contract and the current version actually allows users to set the allowance to an arbitrary value. This had been addressed in the patched `XSwap` contract.

```
    function enableLending(address _token) public auth returns (bool) {
        require(!supportLending[_token], "the token is already supported lending");
        supportLending[_token] = true;

        if (_token == dai) {
            IERC20Token(_token).approve(chai, uint256(-1));
            IERC20Token(chai).approve(lendFMe, uint256(-1));
        }
        else {
            NonStandardIERC20Token(_token).approve(lendFMe, uint256(-1));
        }
```

```

        uint256 _balance = NonStandardIERC20Token(_token).balanceOf(address(this));
        if(_balance > 0) {
            if (_token == dai) {
                IChai(chai).join(address(this), _balance);
                ILendFMe(lendFMe).supply(chai, IERC20Token(chai).balanceOf(address(this)
                    ));
            }
            else
                ILendFMe(lendFMe).supply(_token, _balance);
        }
        return true;
    }
```

Listing 3.10: contracts/XSwap.sol

3.6 Excessive Fallback Logic

- ID: PVE-006
- Severity: Informational
- Likelihood: None
- Impact: None
- Target: `contracts/upgradeability/BaseAdminUpgradeabilityProxy.sol`
- Category: Coding Practices [7]
- CWE subcategory: CWE-561 [5]

Description

```
  modifier ifAdmin() {
    if (msg.sender == _admin()) {
      _;
    } else {
      _fallback();
    }
  }
```

Listing 3.11: contracts/upgradeability/BaseAdminUpgradeabilityProxy.sol

Recommendation Remove the `_fallback()` call.

```
  modifier ifAdmin() {
    if (msg.sender == _admin()) {
      _;
    }
  }
```

Listing 3.12: contracts/upgradeability/BaseAdminUpgradeabilityProxy.sol

3.7 Business Logic Issue in trade()

- ID: PVE-007
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `contracts/XSwap.sol`
- Category: Behavioral Problems [8]
- CWE subcategory: CWE-841 [6]

Description

When users call `trade()` with a `_output` token which is in `supportLending[]`, the amount of `_output` token inside LendFMe would be withdrew to complete the trading (line 81 and 85). However, if LendFMe does not have enough amount witheld, the withdrawal simply reverts even when the `XSwap` contract has some token balance which could happen when someone transferred `_output` tokens into the `XSwap` contract directly.

```
60    function trade(address _input, address _output, uint256 _inputAmount, address
          _receiver) public returns (bool) {
61        require(isOpen, "not open");
62        require(prices[_input][_output] != 0, "invalid token address");
63        require(decimals[_input] != 0, "input decimal not setteled");
64        require(decimals[_output] != 0, "output decimal not setteled");
65
66        NonStandardIERC20Token(_input).transferFrom(msg.sender, address(this),
              _inputAmount);
67        if(supportLending[_input]) {
68            if (_input == dai) {
69                IChai(chai).join(address(this), _inputAmount);
70                ILendFMe(lendFMe).supply(chai, IERC20Token(chai).balanceOf(address(this)
                      ));
71            }
72            else
73                ILendFMe(lendFMe).supply(_input, _inputAmount);
74        }
75        uint256 _tokenAmount = normalizeToken(_input, _inputAmount).mul(prices[_input][
              _output]).div(OFFSET);
76        uint256 _fee = _tokenAmount.mul(fee[_input][_output]).div(OFFSET);
77        uint256 _amountToUser = _tokenAmount.sub(_fee);
78
79        if(supportLending[_output]) {
80            if (_output == dai) {
81                ILendFMe(lendFMe).withdraw(chai, _amountToUser); //assume chai / dai >=
                      1;
82                IChai(chai).draw(address(this), _amountToUser);
83            }
84            else
85                ILendFMe(lendFMe).withdraw(_output, denormalizedToken(_output,
                      _amountToUser));
```

```
        }
        NonStandardIERC20Token(_output).transfer(_receiver, denormalizedToken(_output,
            _amountToUser));
        return true;
    }
```

Listing 3.13: contracts/XSwap.sol

Recommendation Utilize the token balance of XSwap contract for trading.

3.8 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 0.5.4;` instead of `pragma solidity ^0.5.4;`.

In addition, there is a known compiler issue that in all 0.5.x solidity prior to `Solidity 0.5.17`. Specifically, a private function can be overridden in a derived contract by a private function of the same name and types. Fortunately, there is no overriding issue in this code, but we still recommend using `Solidity 0.5.17` or above.

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

In this audit, we thoroughly analyzed the X-Swap documentation and implementation. The audited system does involve various intricacies in both design and implementation. The current code base is well organized and those identified issues are promptly confirmed and fixed.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

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 [14, 15, 16, 17, 19].
- Result: Not found
- Severity: Critical

5.1.5 Reentrancy

- Description: Reentrancy [20] 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] MITRE. CWE-190: Integer Overflow or Wraparound. https://cwe.mitre.org/data/definitions/190.html.

[3] MITRE. CWE-248: Uncaught Exception. https://cwe.mitre.org/data/definitions/248.html.

[4] MITRE. CWE-280: Improper Handling of Insufficient Permissions or Privileges. https://cwe.mitre.org/data/definitions/280.html.

[5] MITRE. CWE-561: Dead Code. https://cwe.mitre.org/data/definitions/561.html.

[6] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

[7] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[8] MITRE. CWE CATEGORY: Behavioral Problems. https://cwe.mitre.org/data/definitions/438.html.

[9] MITRE. CWE CATEGORY: Error Conditions, Return Values, Status Codes. https://cwe.mitre.org/data/definitions/389.html.

[10] MITRE. CWE CATEGORY: Numeric Errors. https://cwe.mitre.org/data/definitions/189.html.

[11] MITRE. CWE CATEGORY: Privilege Issues. https://cwe.mitre.org/data/definitions/265.html.

[12] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

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

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

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

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

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

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

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

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