SwanChain Market Providers

SMART CONTRACT AUDIT REPORT

December 2024

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by SwanChain Market Providers to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- 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: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into for: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly, Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood \ IMPACT	Informational	Low	Medium	High
High	Informational	Medium	High	Critical
Medium	Informational	Low	Medium	High
Low	Informational	Low	Low	Medium

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or

analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment 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 1.2.

- 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.
- Code and business security testing: 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.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft Fail Attack
	Hard Fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control
	System API Call Analysis
	Contract Deployment Consistency Check

Category	Assessment Item
Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

2.1 Project Info And Contract Address

Project Name: SwanChain Market Providers

Audit Time: December 2nd, 2024 – December 10th, 2024

Language: Solidity

File Name	HASH
SwanChain Market Providers	https://github.com/swanchain/market-providers/commit/77b78e5c06ca1dd816ecde294f619c5361b61838
SwanChain Market Providers	https://github.com/swanchain/market-providers/commit/e2328f77fb1efb069a20a06c19d9f2253354616a

2.2 Summary

Severity	Found	
Critical	0	
High	0	
Medium	1	
Low	4	
Informational	3	

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE-001	Medium	Failure to determine the slashAmount value will result in balances[cp] being negative, and ultimately lead to a large amount of funds being transferred from the contract	Fixed	Confirmed
NVE-002	Low	May be registered by any address	Ignore	Confirmed
NVE-003	Low	Repeated requests using the requestWithdraw method will overwrite the previous result	Ignore	Confirmed
NVE-004	Low	The multiAddresses array does not determine the uniqueness of the address	Ignore	Confirmed
NVE-005	Low	Privileged roles	Relieved	Confirmed
NVE-006	Informational	Unused code	Fixed	Confirmed
NVE-007	Informational	No check on the amount sent	Ignore	Confirmed
NVE-008	Informational	There is no check whether the address parameter passed into the method is address(0)	Fixed	Confirmed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Failure to determine the slashAmount value will result in balances[cp] being negative, and ultimately lead to a large amount of funds being transferred from the contract.

ID:	NVE-001	Location:	zk-engine/contract/ECPCollateral.sol zk-engine/contract/ZKSequencer.sol
Severity:	Medium	Category:	Business Issues
Likelihood:	Low	Impact:	High

Description:

In the slashCollateral method in the contract, slashAmount is deducted from the frozen balance (frozenBalance) and the normal balance (balances), and slashedFunds increases by slashAmount regardless of whether the balance is sufficient: slashedFunds += slashAmount;

If slashAmount exceeds the actual available funds of the account (the sum of the frozen balance and the normal balance), slashedFunds will be incorrectly increased because there is no limit on the value of fromBalance. This means that even if the account does not have enough balance to support this slashing operation, slashedFunds will increase by an excessively high value. If slashedFunds is incorrectly increased, it may eventually cause fund security issues in the contract. The slashedFunds variable does not properly check the updated value, resulting in the administrator being able to withdraw an amount that exceeds the actual frozen funds or available balance of the contract in withdrawSlashedFunds.

(The same problem exists with the ZKSequencer contract.)

```
92      function slashCollateral(address cp, uint slashAmount) public onlyAdmin {
93          uint availableFrozen = frozenBalance[cp];
94          uint fromFrozen = slashAmount > availableFrozen ? availableFrozen : slashAmount;
95          uint fromBalance = slashAmount > fromFrozen ? slashAmount - fromFrozen : 0;
96
97          frozenBalance[cp] -= fromFrozen;
98          balances[cp] -= int(fromBalance);
99          slashedFunds += slashAmount;
100         checkCpInfo(cp);
101         emit CollateralSlashed(cp, slashAmount);
102         emit CollateralAdjusted(cp, fromFrozen, fromBalance, "Slashed");
103     }
```

```
256     function withdrawSlashedFunds(uint slashfund) public onlyOwner {
257         require(slashedFunds >= slashfund, "Withdraw slashfund amount exceeds slashedFunds");
258         slashedFunds -= slashfund;
259         collateralToken.transfer(msg.sender, slashfund);
260         emit WithdrawSlash(msg.sender, slashfund);
261     }
```

Recommendations:

Exvul Web3 Security recommends limiting the maximum value of slashedFunds. In the slashCollateral method, slashedFunds should be limited to only increase the effective penalty amount, and cannot be increased too much through unreasonable operations. Check the withdrawal conditions of slashedFunds. In the withdrawSlashedFunds method, the withdrawal conditions of slashedFunds should be further restricted to ensure that only the contract actually has enough assets to support the withdrawal operation. At the same time, set the contract to have enough assets to support the withdrawal operation in other places where the contract withdraws funds. It is also recommended to use int() with caution.

Result: Confirmed

Fix Result: Fixed

Added a check that the current value will not be exceeded.

Fixed version: e2328f77fb1efb069a20a06c19d9f2253354616a

3.2 May be registered by any address

ID:	NVE-002	Location:	zk-engine/contract/TaskRegister.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

The registerTaskContract method in the TaskRegistry contract is used to register contracts, but since there is no permission control, any address can call registerTaskContract to register any task contract to the contract. This may cause malicious users to register fake task contracts or bind illegal task contracts to a certain owner.

```
13      function registerTaskContract(address task, address owner) external {
14          require(task != address(0), "Invalid task contract address");
15          require(owner != address(0), "Invalid owner address");
16          require(taskContracts[task].taskContract == address(0), "Task contract already registered");
17
18          taskContracts[task] = TaskContractInfo({
19              owner: owner,
20              taskContract: task
21          });
22
23          emit TaskContractRegistered(task, owner);
24      }
25  }
```

Recommendations:

Exvul Web3 Security recommends adding permission controls (such as onlyOwner or onlyAdmin modifiers) to the registerTaskContract function to ensure that only the owner or administrator of the contract can register a task contract.

Result: Confirmed

Fix Result: Ignore

Customer response: This contract is a common address that every cp worker needs to call. Any address call will not affect the actual business logic. The business logic judgment is based on the task content itself. The call here is just to facilitate scanning all submitted tasks.

3.3 Repeated requests using the requestWithdraw method will overwrite the previous result

ID:	NVE-003	Location:	zk-engine/contract/ECPCollateral.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

The requestWithdraw method will overwrite the last withdrawal request. Each time a user requests a withdrawal, the new withdrawal request will overwrite the old request. Therefore, if the user calls this method multiple times, the previous withdrawal request will be replaced by the new request, thus overwriting the previous request information.

```
131     function requestWithdraw(address cpAccount, uint amount) public {
132         (bool success, bytes memory CPOwner) = cpAccount.call(abi.encodeWithSignature("getOwner()"));
133         require(success, "Failed to call getOwner function of CPAccount");
134         address cpOwner = abi.decode(CPOwner, (address));
135         require(msg.sender == cpOwner, "Only CP's owner can request withdrawal");
136 
137         require(frozenBalance[cpAccount] >= amount, "Not enough frozen balance to withdraw");
138 
139         withdrawRequests[cpAccount] = WithdrawRequest({
140             amount: amount,
141             requestBlock: block.number
142         });
143 
144         emit WithdrawRequested(cpAccount, amount);
145     }
```

Recommendations:

Exvul Web3 Security recommends that you consider providing multiple request records for each user or performing additional checks to prevent multiple requests from being overwritten. Or add a timestamp to the withdrawal request to ensure that users do not overwrite requests frequently.

Result: Confirmed

Fix Result: Ignore

Customer response:

This is our intentional design to only allow one request to exist.

3.4 The multiAddresses array does not determine the uniqueness of the address

ID:	NVE-004	Location:	computing-provider/account/CPAccount.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

In the CPAccount contract, the changeMultiaddrs method can be used to modify the multiAddresses array and is called by the privileged role owner. Currently, the contract does not reflect where the multiAddresses array is used, but there may be such a problem: the array name can be used to determine that this is a multi-address array, which may be used for fund distribution or other judgments, but if multiple addresses in the address array here are the same, there may be security risks.

```
91      function changeMultiaddrs(string[] memory newMultiaddrs) public onlyOwner {
92          emit MultiaddrsChanged(multiAddresses, newMultiaddrs);
93          multiAddresses = newMultiaddrs;
94      }
```

Recommendations:

Exvul Web3 Security recommends setting address uniqueness in the multiAddresses array.

Result: Confirmed

Fix Result: Ignore

Customer response:

This is to reserve space for ipv6 addresses. Multi-address is not the only identifier of cp, cpAccount is the only identifier.

3.5 Privileged roles

ID:	NVE-005	Location:	computing-provider/account/CPAccount.sol zk-engine/contract/ECPCollateral.sol zk-engine/contract/ZKSequencer.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

The owner privileged role is used in many places in the project. The privileged role has almost all permissions, including fund management and contract parameter settings. If the privileged account is stolen or malicious behavior occurs, the assets of the entire contract may be extracted or the parameters may be tampered with.

```
215     function setCollateralToken(address tokenAddress) external onlyOwner {
216         collateralToken = IERC20(tokenAddress);
217     }
218
219     function setCollateralRatio(uint _collateralRatio) external onlyOwner {
220         collateralRatio = _collateralRatio;
221     }
222
223     function setSlashRatio(uint _slashRatio) external onlyOwner {
224         slashRatio = _slashRatio;
225     }
226
227     function setBaseCollateral(uint _baseCollateral) external onlyAdmin {
228         baseCollateral = _baseCollateral;
229     }
230
231     function setWithdrawDelay(uint _withdrawDelay) external onlyOwner {
232         withdrawDelay = _withdrawDelay;
233     }
234
235     function getBaseCollateral() external view returns (uint) {
236         return baseCollateral;
237     }
238
239         function cpInfo(address cpAccount) external view returns (CPInfo memory) {
240         return CPInfo({
241             cp: cpAccount,
242             balance: balances[cpAccount],
243             frozenBalance: frozenBalance[cpAccount],
244             status: cpStatus[cpAccount]
245         });
246     }
247
248     function checkCpInfo(address cpAccount) internal {
249         if (balances[cpAccount] == 0 && frozenBalance[cpAccount] == 0) {
250             cpStatus[cpAccount] = "NSC";
251         } else {
252             cpStatus[cpAccount] = "Active";
253         }
254     }
255
256     function withdrawSlashedFunds(uint slashfund) public onlyOwner {
257         require(slashedFunds >= slashfund, "Withdraw slashfund amount exceeds slashedFunds");
258         slashedFunds -= slashfund;
259         collateralToken.transfer(msg.sender, slashfund);
260         emit WithdrawSlash(msg.sender, slashfund);
261     }
```

Recommendations:

Exvul Web3 Security recommends that privileged roles be managed using multi-signatures.

Result: Confirmed

Fix Result: Relieved

Customer response:

Multi-sign wallet is necessary and will be supported in subsequent versions. The current version only uses the basic operation method of the admin and the owner offline storage to ensure security.

3.6 Unused code

ID:	NVE-006	Location:	computing-provider/account/CPAccount.sol
Severity:	Informational	Category:	Business Issues
Likelihood:	Low	Impact:	Informational

Description:

In the CPAccount contract, the ownerAndWorker modifier is mainly used to determine whether the caller is the owner or the worker, but this modifier is not used. It is recommended to delete it.

```
51    modifier ownerAndWorker() {
52        require(msg.sender == owner|| msg.sender == worker, "owner and worker can call this function.");
53        _;
54    }
```

Recommendations:

Exvul Web3 Security recommends removing unused code.

Result: Confirmed

Fix Result: Fixed

Redundant code has been deleted.

Fixed version: e2328f77fb1efb069a20a06c19d9f2253354616a

3.7 No check on the amount sent

ID:	NVE-007	Location:	zk-engine/contract/ZKSequencer.sol zk-engine/contract/ECPCollateral.sol
Severity:	Informational	Category:	Business Issues
Likelihood:	Low	Impact:	Informational

Description:

In the requestWithdraw, deposit, and withdraw methods, there is no explicit check for whether amount is zero.

deposit: Users can try to deposit zero collateral (although most ERC20 token transfers will fail, the contract should perform an explicit check).

withdraw: Users may try to withdraw zero, which not only wastes gas fees but may also cause logic problems.

requestWithdraw: Users can request a withdrawal of zero, resulting in unnecessary requests being processed.

```
40      function deposit(address cpAccount) public payable {
41          require(cpAccount != address(0), "Invalid account address");
42          balances[cpAccount] += int(msg.value);
43          emit Deposited(cpAccount, msg.value);
44      }
```

Recommendations:

Exvul Web3 Security recommends setting a minimum value for the incoming parameters to avoid abuse.

Result: Confirmed

Fix Result: Ignore

3.8 There is no check whether the address parameter passed into the method is address(0)

ID:	NVE-008	Location:	zk-engine/contract/ZKSequencer.sol zk-engine/contract/ECPCollateral.sol
Severity:	Informational	Category:	Business Issues
Likelihood:	Low	Impact:	Informational

Description:

Multiple methods in the ZKSequencer and ECPCollateral contract do not perform zero address judgment on the passed address parameters, which can cause a variety of impacts, such as poor user experience or unexpected rollbacks, freezing zero address balances, or causing contract logic exceptions.

```
47      function withdraw(address cpAccount, uint256 amount) external {
48          (bool success, bytes memory CPOwner) = cpAccount.call(abi.encodeWithSignature("getOwner()"));
49          require(success, "Failed to call getOwner function of CPAccount");
50          address cpOwner = abi.decode(CPOwner, (address));
51          require(balances[cpAccount] >= int(amount), "Withdraw amount exceeds balance");
52          require(msg.sender == cpOwner, "Only CP's owner can withdraw the collateral funds");
53          balances[cpAccount] -= int(amount);
54          payable(msg.sender).transfer(amount);
55          emit Withdrawn(cpAccount, amount);
56      }
57
58      function transferToEscrow(address cpAccount, uint256 amount) external onlyAdminOrOwner {
59          balances[cpAccount] -= int(amount);
60          escrowBalance += amount;
61          emit TransferredToEscrow(cpAccount, amount);
62      }
```

Recommendations:

Exvul Web3 Security recommends adding zero address verification and adding checks at the method entry to prevent the introduction of zero addresses.

Result: Confirmed

Fix Result: Fixed

Zero address check has been added.

Fixed version: e2328f77fb1efb069a20a06c19d9f2253354616a

```
58      function transferToEscrow(address cpAccount, uint256 amount) external onlyAdminOrOwner {
59          require(cpAccount != address(0), "Invalid address: cpAccount cannot be the zero address");
```

4. CONCLUSION

In this audit, we thoroughly analyzed **SwanChain Market Providers** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be Passed. 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 Assessment

5.1.1 Apply Verification Control

- Description: The security of apply verification
- Result: Not found
- Severity: Critical

5.1.2 Authorization Access Control

- Description: Permission checks for external integral functions
- Result: Not found
- Severity: Critical

5.1.3 Forged Transfer Vulnerability

- Description: Assess whether there is a forged transfer notification vulnerability in the contract
- Result: Not found
- Severity: Critical

5.1.4 Transaction Rollback Attack

- Description: Assess whether there is transaction rollback attack vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.5 Transaction Block Stuffing Attack

- Description: Assess whether there is transaction blocking attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.6 Soft Fail Attack Assessment

- Description: Assess whether there is soft fail attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.7 Hard Fail Attack Assessment

- Description: Examine for hard fail attack vulnerability
- Result: Not found
- Severity: Critical

5.1.8 Abnormal Memo Assessment

- Description: Assess whether there is abnormal memo vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.9 Abnormal Resource Consumption

- Description: Examine whether abnormal resource consumption in contract processing.
- Result: Not found
- Severity: Critical

5.1.10 Random Number Security

- Description: Examine whether the code uses insecure random number.
- Result: Not found
- Severity: Critical

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

- Description: Examine for weakness in cryptograph implementation.
- Results: Not Found
- Severity: High

5.2.2 Account Permission Control

- Description: Examine permission control issue in the contract
- Results: Not Found
- Severity: Medium

5.2.3 Malicious Code Behavior

- Description: Examine whether sensitive behavior present in the code
- Results: Not found
- Severity: Medium

5.2.4 Sensitive Information Disclosure

- Description: Examine whether sensitive information disclosure issue present in the code.
- Result: Not found
- Severity: Medium

5.2.5 System API

- Description: Examine whether system API application issue present in the code
- Results: Not found
- Severity: Low

6. DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

7. REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound).

https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error.

https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption.

https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation.

https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure.

https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features.

https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems.

https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors.

https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors.

https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology.

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

www.exvul.com

contact@exvul.com

@EXVULSEC

github.com/EXVUL-Sec

ExVul