SMART CONTRACT AUDIT REPORT

for

Multichain

Prepared By: Patrick Lou

PeckShield

March 17, 2022

Document Properties

Client	Multichain
Title	Smart Contract Audit Report
Target	Multichain
Version	1.0
Author	Shulin Bie
Auditors	Shulin Bie, Xuxian Jiang
Reviewed by	Patrick Lou
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	March 17, 2022	Shulin Bie	Final Release
1.0-rc	March 12, 2022	Shulin Bie	Release Candidate

Contact

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

Name	Patrick Lou
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the Multichain, we outline in the report 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 Multichain

Multichain aims to be the ultimate router for Web3. It is an infrastructure developed for arbitrary cross-chain interactions. Arguably one of the largest cross-chain protocols in terms of Total Value Locked (TVL) and liquidity, it has so far supported almost all blockchains for interaction and crosschain transfers, including Ethereum, BNB Chain (previously BSC), Polygon, Avalanche, Fantom, etc. Multichain enriches the DeFi market and also presents a unique contribution to current DeFi ecosystem.

Table 1.1: Basic Information of Multichain

Item	Description
Target	Multichain
Type	Smart Contract
Language	Solidity
Audit Method	Whitebox
Latest Audit Report	March 17, 2022

In the following, we show the Git repositories of reviewed files and the commit hash values used in this audit. Please note this audit only covers the following contract files in these two Git repositories: AnyswapV6ERC20.sol, AnyswapV6Router.sol, AnyswapV4CallProxy.sol, MultichainToken.sol, MultiDAO.sol, and SwapTokens.sol.

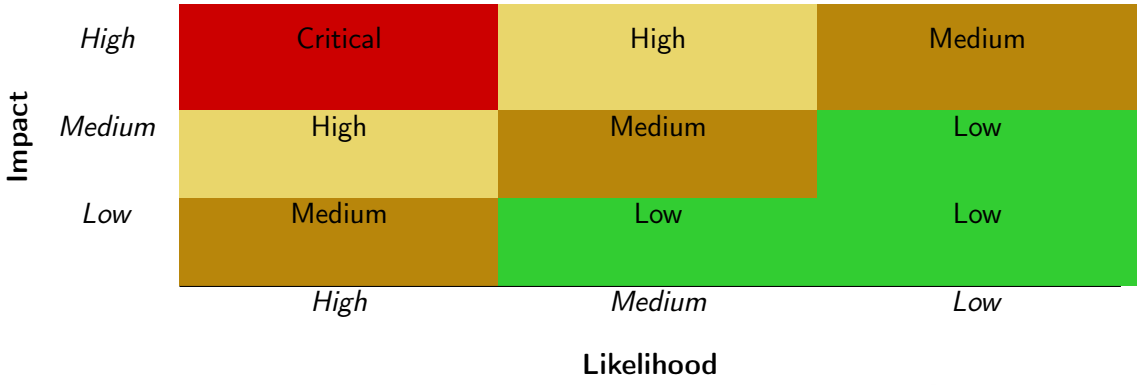
- https://github.com/anyswap/anyswap-v1-core.git (f0c62f8)

- https://github.com/anyswap/Multichain-token.git (20372be)

1.2 About PeckShield

PeckShield Inc. [11] 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



1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10]:

- 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

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

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.

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) [9], 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 security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment 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 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 `Multichain` implementation. During the first phase of our audit, we study the smart contract source code and run 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 logic, 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	1	■
Low	1	■
Informational	3	■■■
Total	5	

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 that 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 1 medium-severity vulnerability, 1 low-severity vulnerability, and 3 informational recommendations.

Table 2.1: Key Multichain Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Informational	Improved Validation Of Function Arguments	Coding Practices	Confirmed
PVE-002	Informational	Suggested Event Generation For Key Operations	Coding Practices	Confirmed
PVE-003	Informational	Suggested Fine-Grained Risk Control Of Transfer Volume	Security Features	Mitigated
PVE-004	Medium	Trust Issue Of Admin Keys	Security Features	Mitigated
PVE-005	Low	Fork-Compliant Domain Separator in AnyswapV6ERC20	Business Logic	Confirmed

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 Detailed Results

3.1 Improved Validation Of Function Arguments

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `AnyswapV6Router`
- Category: Coding Practices [7]
- CWE subcategory: CWE-1041 [1]

Description

According to the `Multichain` design, the `AnyswapV6Router` contract is designed to be the main entry for cross-chain transactions. In particular, one routine, i.e., `anySwapOut()`, combines the multiple cross-chain asset transfers into one transaction. While examining its logic, we observe the current implementation can be improved.

To elaborate, we show below the related code snippet of the contract. In the `anySwapOut()` function, we notice it has the inherent assumption on the same length of the given four arrays, i.e., `tokens`, `to`, `amounts`, and `toChainIDs`. However, this is not enforced inside the `anySwapOut()` function. For improvement, it is helpful to validate the length of these four arrays.

```
322     function anySwapOut(address[] calldata tokens, address[] calldata to, uint[]
            calldata amounts, uint[] calldata toChainIDs) external {
323         for (uint i = 0; i < tokens.length; i++) {
324             _anySwapOut(msg.sender, tokens[i], to[i], amounts[i], toChainIDs[i]);
325         }
326     }
```

Listing 3.1: `AnyswapV6Router::anySwapOut()`

Note that another routine, i.e., `anySwapIn()`, can be similarly improved.

Recommendation Validate the length of the input arrays in the above-mentioned routines.

Status The issue has been confirmed.

3.2 Suggested Event Generation For Key Operations

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Multiple Contracts`
- Category: Coding Practices [7]
- CWE subcategory: CWE-563 [3]

Description

In `Ethereum`, the `event` is an indispensable part of a contract and is mainly used to record a variety of runtime dynamics. In particular, when an `event` is emitted, it stores the arguments passed in transaction logs and these logs are made accessible to external analytics and reporting tools. `Events` can be emitted in a number of scenarios. One particular case is when system-wide parameters or settings are being changed. Another case is when tokens are being minted, transferred, or burned.

While examining the events that reflect the protocol dynamics, we notice there are several privileged routines that lack meaningful events to reflect their changes. In the following, we show several representative routines.

```
190     function setVault(address _vault) external onlyVault {
191         require(_vault != address(0), "AnyswapV3ERC20: address(0x0)");
192         pendingVault = _vault;
193         delayVault = block.timestamp + delay;
194     }
195
196     function applyVault() external onlyVault {
197         require(block.timestamp >= delayVault);
198         vault = pendingVault;
199     }
200
201     function setMinter(address _auth) external onlyVault {
202         require(_auth != address(0), "AnyswapV3ERC20: address(0x0)");
203         pendingMinter = _auth;
204         delayMinter = block.timestamp + delay;
205     }
206
207     function applyMinter() external onlyVault {
208         require(block.timestamp >= delayMinter);
209         isMinter[pendingMinter] = true;
210         minters.push(pendingMinter);
211     }
212
213     // No time delay revoke minter emergency function
214     function revokeMinter(address _auth) external onlyVault {
215         isMinter[_auth] = false;
```

```
216     }
```

Listing 3.2: AnyswapV6ERC20

With that, we suggest to emit meaningful events in these privileged routines. Also, the key event information is better indexed. Note each emitted event is represented as a topic that usually consists of the signature (from a keccak256 hash) of the event name and the types (uint256, string, etc.) of its parameters. Each indexed type will be treated like an additional topic. If an argument is not indexed, it will be attached as data (instead of a separate topic). Considering that the key information is typically queried, it is better treated as a topic, hence the need of being indexed.

Note the other routines, i.e., MultiDao::transferOwnership(), SwapTokens::transferOwnership(), and AnyswapV6Router::setEnableSwapTrade(), also lack meaningful events.

Recommendation Properly emit the above-mentioned events with accurate information to timely reflect state changes. This is very helpful for external analytics and reporting tools.

Status The issue has been confirmed.

3.3 Suggested Fine-Grained Risk Control Of Transfer Volume

- ID: PVE-003
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: AnyswapV6Router/AnyswapV6ERC20
- Category: Security Features [6]
- CWE subcategory: CWE-654 [4]

Description

According to the Multichain design, each deployed AnyswapV6ERC20 contract will likely accumulate a huge amount of underlying tokens with the increased popularity of cross-chain transactions. While examining the current implementation of the AnyswapV6Router/AnyswapV6ERC20 contracts, we notice there is no risk control based on the requested transfer amount, including but not limited to, daily transfer volume restriction and per-transaction transfer volume restriction. This is reasonable under the assumption that the protocol will always work well without any vulnerability and the MPC key is always properly managed. In the following, we take the AnyswapV6Router::anySwapOut()/anySwapIn() routines to elaborate our suggestion.

Specifically, we show below the related code snippet of the AnyswapV6Router contract. According to the Multichain design, when the anySwapOut() function is called on the source chain, the anySwapIn() function on the destination chain will be called subsequently by the privileged MPC account to mint the bridge token to the recipient as cross-chain transfer credits. In the anySwapIn() function, we

notice the only protection is validating the msg.sender. If we assume the privileged MPC account is hijacked or leaked, all the assets locked up in the AnyswapV6ERC20 contract will be in risk. To mitigate, we suggest to add fine-grained risk controls based on the requested transfer volume. A guarded launch process is also highly recommended.

```
346     function _anySwapIn(bytes32 txs, address token, address to, uint amount, uint
            fromChainID) internal {
347         AnyswapV1ERC20(token).mint(to, amount);
348         emit LogAnySwapIn(txs, token, to, amount, fromChainID, cID());
349     }
350
351     // swaps `amount` `token` in `fromChainID` to `to` on this chainID
352     // triggered by `anySwapOut`
353     function anySwapIn(bytes32 txs, address token, address to, uint amount, uint
            fromChainID) external onlyMPC {
354         _anySwapIn(txs, token, to, amount, fromChainID);
355     }
```

Listing 3.3: AnyswapV6Router::anySwapIn()

Recommendation We suggest to add fine-grained risk controls, including but not limited to daily transfer volume restriction and per transaction transfer volume restriction.

Status The issue has been confirmed. The Multichain bridge backend has the transfer volume restriction in place for the same purpose.

3.4 Trust Issue Of Admin Keys

- ID: PVE-004
- Severity: Medium
- Likelihood: Low
- Impact: High
- Target: Multiple Contracts
- Category: Security Features [6]
- CWE subcategory: CWE-287 [2]

Description

In the Multichain protocol, there is a privileged MPC account that plays a critical role in governing and regulating the protocol-wide operations (e.g., transfer the locked underlying token out of the contract). In the following, we show the representative functions potentially affected by the privilege of the account.

```
395     // extracts mpc fee from bridge fees
396     function anySwapFeeTo(address token, uint amount) external onlyMPC {
397         address _mpc = mpc();
398         AnyswapV1ERC20(token).mint(_mpc, amount);
399         AnyswapV1ERC20(token).withdrawVault(_mpc, amount, _mpc);
```

```
400     }
```

Listing 3.4: AnyswapV6Router::anySwapFeeTo()

```
201     function setMinter(address _auth) external onlyVault {
202         require(_auth != address(0), "AnyswapV3ERC20: address(0x0)");
203         pendingMinter = _auth;
204         delayMinter = block.timestamp + delay;
205     }
206
207     function applyMinter() external onlyVault {
208         require(block.timestamp >= delayMinter);
209         isMinter[pendingMinter] = true;
210         minters.push(pendingMinter);
211     }
```

Listing 3.5: AnyswapV6ERC20::setMinter()&&applyMinter()

We emphasize that the privilege assignment is indeed necessary and consistent with the protocol design. However, it is worrisome if the privileged account is a plain EOA account. A multi-sig account could greatly alleviate this concern, though it is still far from perfect. Note that a compromised privileged account would allow the attacker to modify a number of sensitive system parameters, which directly undermines the assumption of the protocol design.

Recommendation Suggest a multi-sig account plays the privileged account to mitigate this issue. Additionally, all changes to privileged operations may need to be mediated with necessary timelocks.

Status This issue has been mitigated as the team confirms that the privileged MPC account is effectively a multi-sig account.

3.5 Fork-Compliant Domain Separator in AnyswapV6ERC20

- ID: PVE-005
- Severity: Low
- Likelihood: Low
- Impact: High
- Target: AnyswapV6ERC20
- Category: Business Logic [8]
- CWE subcategory: CWE-841 [5]

Description

The AnyswapV6ERC20 token contract strictly follows the widely-accepted ERC20 specification. In the meantime, we notice the support of EIP-2612 with the permit() function that allows for approvals to be made via secp256k1 signatures. Interestingly, we notice the state variable DOMAIN_SEPARATOR is initialized once inside the constructor() function (lines 287-293).

```
266     constructor(string memory _name, string memory _symbol, uint8 _decimals, address
            _underlying, address _vault) {
267         name = _name;
268         symbol = _symbol;
269         decimals = _decimals;
270         underlying = _underlying;
271         if (_underlying != address(0x0)) {
272             require(_decimals == IERC20(_underlying).decimals());
273         }
274
275         // Use init to allow for CREATE2 accross all chains
276         _init = true;
277
278         // Disable/Enable swapout for v1 tokens vs mint/burn for v3 tokens
279         _vaultOnly = false;
280
281         vault = _vault;
282         pendingVault = _vault;
283         delayVault = block.timestamp;
284
285         uint256 chainId;
286         assembly {chainId := chainid()}
287         DOMAIN_SEPARATOR = keccak256(
288             abi.encode(
289                 keccak256("EIP712Domain(string name,string version,uint256 chainId,
                        address verifyingContract)"),
290                 keccak256(bytes(name)),
291                 keccak256(bytes("1")),
292                 chainId,
293                 address(this)));
294     }
```

Listing 3.6: AnyswapV6ERC20::constructor()

The DOMAIN_SEPARATOR is used in the permit() function and should be unique to the contract and chain in order to prevent replay attacks from other domains. However, when analyzing this permit() routine, we realize the current implementation needs to be improved by recalculating the value of DOMAIN_SEPARATOR inside the permit() function, for the very purpose of preventing cross-chain replay attacks. Specifically, when there is a chain-level hard-fork, because of the pre-computed DOMAIN_SEPARATOR, a valid signature for one chain could be replayed on the other. Note the same issue is also applicable to another routine, i.e, transferWithPermit().

```
418     function permit(address target, address spender, uint256 value, uint256 deadline,
            uint8 v, bytes32 r, bytes32 s) external override {
419         require(block.timestamp <= deadline, "AnyswapV3ERC20: Expired permit");
420
421         bytes32 hashStruct = keccak256(
422             abi.encode(
423                 PERMIT_TYPEHASH,
424                 target,
425                 spender,
```

```
426                 value,
427                 nonces[target]++,
428                 deadline));
429
430         require(verifyEIP712(target, hashStruct, v, r, s)  verifyPersonalSign(target,
                hashStruct, v, r, s));
431
432         // _approve(owner, spender, value);
433         allowance[target][spender] = value;
434         emit Approval(target, spender, value);
435     }
```

Listing 3.7: `AnyswapV6ERC20::permit()`

Recommendation Recalculate the value of `DOMAIN_SEPARATOR` inside the `permit()` function.

Status The issue has been confirmed.

4 Conclusion

In this audit, we have analyzed the Multichain design and implementation. Multichain aims to be the ultimate router for Web3 and by design is an infrastructure developed for arbitrary cross-chain interactions. Arguably one of the largest cross-chain protocols in terms of Total Value Locked (TVL) and liquidity, it has so far supported almost all blockchains to inter-operate. Multichain greatly enriches the DeFi market and also presents a unique contribution to current DeFi ecosystem. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

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.

References

[1] MITRE. CWE-1041: Use of Redundant Code. https://cwe.mitre.org/data/definitions/1041.html.

[2] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[3] MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

[4] MITRE. CWE-654: Reliance on a Single Factor in a Security Decision. https://cwe.mitre.org/data/definitions/654.html.

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

[6] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

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

[8] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

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

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

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