# SMART CONTRACT AUDIT REPORT

Mango Smart Contract

JULY 2025

## Contents

**1. EXECUTIVE SUMMARY** 4
1.1 Methodology . . . . . 4

**2. FINDINGS OVERVIEW** 7
2.1 Project Info And Contract Address . . . . . 7
2.2 Summary . . . . . 7
2.3 Key Findings . . . . . 8

**3. DETAILED DESCRIPTION OF FINDINGS** 9
3.1 ID occupied causes program termination . . . . . 9
3.2 Multiple signature validation bypassed when total_signers equals zero . . . . . 11
3.3 Incorrect Display of Bridge Info . . . . . 13
3.4 Existence check problem . . . . . 15
3.5 Misleading Function Name . . . . . 17
3.6 Fee Receiver Zero Address Risk . . . . . 18
3.7 Missing Upper Limit Validation for Percentage Fee . . . . . 19
3.8 Unnecessary Store Capability on Struct . . . . . 20

**4. CONCLUSION** 21

**5. APPENDIX** 22
5.1 Basic Coding Assessment . . . . . 22
5.1.1 Apply Verification Control . . . . . 22
5.1.2 Authorization Access Control . . . . . 22
5.1.3 Forged Transfer Vulnerability . . . . . 22
5.1.4 Transaction Rollback Attack . . . . . 23
5.1.5 Transaction Block Stuffing Attack . . . . . 23
5.1.6 Soft Fail Attack Assessment . . . . . 23
5.1.7 Hard Fail Attack Assessment . . . . . 24
5.1.8 Abnormal Memo Assessment . . . . . 24
5.1.9 Abnormal Resource Consumption . . . . . 24
5.1.10 Random Number Security . . . . . 25
5.2 Advanced Code Scrutiny . . . . . 25
5.2.1 Cryptography Security . . . . . 25
5.2.2 Account Permission Control . . . . . 25
5.2.3 Malicious Code Behavior . . . . . 26
5.2.4 Sensitive Information Disclosure . . . . . 26

5.2.5 System API . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

**6. DISCLAIMER** **27**

**7. REFERENCES** **28**

# 1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **Mango** 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: 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* | INFO | MEDIUM | HIGH | CRITICAL |
| *Medium* | INFO | LOW | MEDIUM | HIGH |
| *Low* | INFO | 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<br>• Authorization Access Control<br>• Forged Transfer Vulnerability<br>• Forged Transfer Notification<br>• Numeric Overflow<br>• Transaction Rollback Attack<br>• Transaction Block Stuffing Attack<br>• Soft Fail Attack<br>• Hard Fail Attack<br>• Abnormal Memo<br>• Abnormal Resource Consumption<br>• Secure Random Number |

| | |
|---|---|
| **Advanced Source Code Scrutiny** | • Asset Security<br>• Cryptography Security<br>• Business Logic Review<br>• Source Code Functional Verification<br>• Account Authorization Control<br>• Sensitive Information Disclosure<br>• Circuit Breaker<br>• Blacklist Control<br>• System API Call Analysis<br>• Contract Deployment Consistency Check<br>• Abnormal Resource Consumption |
| **Additional Recommendations** | • Semantic Consistency Checks<br>• 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 | Audit Time | Language |
|---|---|---|
| Mango | 11/07/2025 - 17/07/2025 | Solidity/Rust/Move |

| Repository | Commit Hash |
|---|---|
| Mango/ | b207f90daa57c1bb2c63a74acc853ac0707be8f8 |

## 2.2 Summary

| Severity | Found |
|---|---|
| CRITICAL | 0 |
| HIGH | 0 |
| MEDIUM | 1 |
| LOW | 3 |
| INFO | 4 |

### 2.3 Key Findings

| Severity | Findings Title | Status |
|---|---|---|
| MEDIUM | ID occupied causes program termination | Fixed |
| LOW | Multiple signature validation bypassed when total_signers equals zero | Fixed |
| LOW | Incorrect Display of Bridge Info | Fixed |
| LOW | Existence check problem | Fixed |
| INFO | Misleading Function Name | Acknowledge |
| INFO | Fee Receiver Zero Address Risk | Fixed |
| INFO | Missing Upper Limit Validation for Percentage Fee | Fixed |
| INFO | Unnecessary Store Capability on Struct | Acknowledge |

**Table 2.3: Key Audit Findings**

## 3. DETAILED DESCRIPTION OF FINDINGS

### 3.1 ID occupied causes program termination

**SEVERITY:** MEDIUM **STATUS:** Fixed

**PATH:**

sui/sources/config.move

**DESCRIPTION:**

The new_bridge_pair function in config.move does not strictly enforce the relationship between the provided id and the internal pair_id counter. As a result, when calling create_special_bridge_pair with an id greater than the current pair_id, and subsequently calling create_bridge_pair, the auto-increment behavior of pair_id may lead to an id collision with an already existing entry. This causes the program to abort due to a duplicate ID.

```
fun new_bridge_pair<T>(
    self: &mut BridgeConfig,
    to_token: String,
    id: u64,
    ctx: &mut TxContext
): u64 {
    assert!(!is_pair_exist<T>(self, to_token), EPairExist);
    assert!(id == 0 || (id > 0 && !is_pair_id_exist(self, id)),
        EPairIdExist);
    if (id == 0) {
        self.pair_id = self.pair_id + 1;
        id = self.pair_id;
    };

    table::add(&mut self.pairs, id, pair);
    id
}
```

**IMPACT:**

- Potential program panic due to id conflicts.
- Unpredictable behavior or denial of service when creating new bridge pairs.

## RECOMMENDATIONS:

Add a while loop to the if statement of the new_bridge_pair function to ensure that when the id exists, it continues to increment until the id is not repeated:

```
// config.move --> new_bridge_pair()
-if (id == 0) {
-     self.pair_id = self.pair_id + 1;
-     id = self.pair_id;
-};
+if (id == 0) {
+     while(true) {
+         self.pair_id = self.pair_id + 1;
+         id = self.pair_id;
+        if(!is_pair_id_exist(self, id)) {
+            break
+        };
+     }
+};
```

### 3.2 Multiple signature validation bypassed when total_signers equals zero

**SEVERITY:** LOW **STATUS:** Fixed

**PATH:**

`solana/programs/mango-bridge/src/instructions/sign.rs`

**DESCRIPTION:**

In the sign instruction solana/programs/mango-bridge/src/instructions/sign.rs, when GlobalConfig.signers contains only zero addresses ([0u8; 20]), the signature validation logic is completely bypassed, allowing any user to create valid ClaimSignature accounts and subsequently extract tokens from vaults.

Two Attack Scenarios Leading to total_signers = 0: **Scenario 1:** Empty Signers During Initialization When admin initializes GlobalConfig with empty signers list (e.g., admin wants to set signers later via update), this results in total_signers = 0.

**Scenario 2:** Signer Update Gap When admin wants to replace all current signers with new ones, they call update_global_signers which first removes all existing signers, then adds new ones. This creates a total_signers = 0 gap.

```
let total_signers = ctx
    .accounts
    .global_config
    .signers
    .iter()
    .filter(|s| **s != [0u8; 20])
    .count();  // Returns 0 when all signers are [0u8; 20]

let required_signatures = (total_signers * 2 + 2) / 3;  // = 0

require!(
    sign_count >= required_signatures as u8,  // 0 >= 0 passes
    CustomErrorCode::InsufficientSignatures
);
```

**IMPACT:**

Complete Fund Drainage: Attackers can extract all token reserves from vaults or mint unlimited tokens when vault has mint authority, causing total loss of bridge assets. Zero-Cost Exploitation: No cryptographic signatures or special permissions required, any user can exploit when total_signers = 0.

## RECOMMENDATIONS:

Ensure Minimum Required Signatures:

```
// sign.rs
-let required_signatures = (total_signers * 2 + 2) / 3;
+let required_signatures = std::cmp::max(1, (total_signers * 2 + 2) / 3);

require!(
    sign_count >= required_signatures as u8,
    CustomErrorCode::InsufficientSignatures
);
```

### 3.3 Incorrect Display of Bridge Info

**SEVERITY:** LOW **STATUS:** Fixed

**PATH:**

sui/sources/config.move

**DESCRIPTION:**

In the config.move file, the pair_list function constructs a list of pairs by iterating through pair_id using a while loop. However, it fails to account for special pairs that may have been created using the create_special_bridge_pair function, where the specified id can be greater than the current pair_id.

```
public(friend) fun pair_list(self: &BridgeConfig): vector<BridgePair> {
    let pairs = vector::empty<BridgePair>();
    let index = 1;
    while (index <= self.pair_id) {
        if (table::contains(&self.pairs, index)) {
            let pair = *table::borrow(&self.pairs, index);
            vector::push_back(&mut pairs, pair);
        };
        index = index + 1;
    };
    pairs
}
```

As a result, those special pairs are omitted during iteration. Ultimately, this leads to an incomplete BridgeInfo being returned in the bridge_info function of bridge.move.

**IMPACT:**

When the frontend retrieves pairs information from BridgeInfo via events, it may miss special pairs whose id is less than pair_id.

**RECOMMENDATIONS:**

Add a vector field named pair_keys to the BridgeConfig struct to keep track of the created pair IDs for easier iteration and lookup:

```
// config.move --> BridgeConfig
struct BridgeConfig has store {
    role: Table<address, u8>,
    pair_id: u64,
    pair_ids: Table<TypeName, Table<String, u64>>,
    pairs: Table<u64, BridgePair>,
+   pair_keys: vector<u64>
}
```

### 3.4 Existence check problem

**SEVERITY:** LOW **STATUS:** Fixed

**PATH:**

sui/sources/treasury.move

**DESCRIPTION:**

In the treasury.move file, the add_treasury_cap function does not check if the same typ already exists in treasuries. If so, it will abort.

```
// treasury.move --> add_treasury_cap()
public(friend) fun add_treasury_cap<T>(self: &mut BridgeTreasury,
    treasury_cap: TreasuryCap<T>) {
     let typ = type_name::get<T>();
     if (!vec_set::contains(&self.typs, &typ)) {
         vec_set::insert(&mut self.typs, typ);
     };
     object_bag::add(&mut self.treasuries, typ, treasury_cap)
}
```

The function directly adds to the ObjectBag without checking if the key already exists, which will cause an abort if the treasury type already exists.

**IMPACT:**

If the operation is not done properly, the program will abort.

**RECOMMENDATIONS:**

Before executing the add function, call the is_treasury_exist function to check:

```
// treasury.move --> add_treasury_cap()
public(friend) fun add_treasury_cap<T>(self: &mut BridgeTreasury,
    treasury_cap: TreasuryCap<T>) {
+    assert!(!is_treasury_exist<T>(self), ErrorTreasuryAlreadyExist);
     let typ = type_name::get<T>();
```

```
    if (!vec_set::contains(&self.typs, &typ)) {
        vec_set::insert(&mut self.typs, typ);
    };
    object_bag::add(&mut self.treasuries, typ, treasury_cap)
}
```

### 3.5 Misleading Function Name

**SEVERITY:** INFO **STATUS:** Acknowledge

**PATH:**

solidity/BridgeConfig.sol

**DESCRIPTION:**

The isTokenNotStandard() function name creates a confusing logical contradiction in the codebase. When this function returns true, indicating a "non-standard" token, the code actually performs standard ERC20 direct calls instead of using the safer TransferHelper methods.

```
if (bc.isTokenNotStandard(message.toToken)) {
    IERC20(message.toToken).transfer(
        message.destination,
        message.amountOut
    );
} else {
    TransferHelper.safeTransfer(
        message.toToken,
        message.destination,
        message.amountOut
    );
}
```

**IMPACT:**

- Code Readability: Developers reading the code will be confused by the apparent logical contradiction

**RECOMMENDATIONS:**

Rename the function to better reflect its actual purpose.

### 3.6 Fee Receiver Zero Address Risk

**SEVERITY:** INFO **STATUS:** Fixed

## PATH:

`solidity/BridgeConfig.sol`

## DESCRIPTION:

The setFeeReceiver function in BridgeConfig.sol does not validate the input receiver address. This allows the owner to set the fee-collecting address to address(0).

```
// contracts/BridgeConfig.sol:127-129
function setFeeReceiver(address receiver) external onlyOwner {
    feeReceiver = receiver; //  No validation, can be set to zero address
}
```

The function directly assigns the input address to the feeReceiver state variable without performing a zero-address check.

## IMPACT:

If the feeReceiver is set to zero address, all collected protocol fees will be transferred to an unrecoverable address, leading to a permanent loss of funds. This could also interrupt bridge operations if fee transfers are a prerequisite for transaction processing.

## RECOMMENDATIONS:

Add a required statement to ensure the receiver address is not the zero address and emit an event to log the change.

### 3.7 Missing Upper Limit Validation for Percentage Fee

**SEVERITY:** INFO **STATUS:** Fixed

**PATH:**

`solidity/BridgeConfig.sol`

**DESCRIPTION:**

The modifyPairFee() function in BridgeConfig.sol lacks upper limit validation for percentage fees (feeType = 0). This allows setting fee values greater than 10000 basis points, which would result in fees exceeding 100% of the transaction amount.

**IMPACT:**

User Experience: Users may be charged unexpectedly high fees without sufficient warning, leading to non-obvious financial loss.

**RECOMMENDATIONS:**

Add upper limit validation for percentage fees.

### 3.8 Unnecessary Store Capability on Struct

**SEVERITY:** INFO

**STATUS:** Acknowledge

#### PATH:

`sui/sources/bridge.move`

#### DESCRIPTION:

The BridgeCap Struct is declared with the store ability. However, the struct is neither nested within other structs that require storage nor transferred across modules. Its usage does not justify the need for the store ability.

```
// bridge.move
struct BridgeCap has key, store {
    id: UID
}
```

#### IMPACT:

The unnecessary store capability could potentially create confusion and might allow unintended usage patterns.

#### RECOMMENDATIONS:

Remove the store capability of the BridgeCap struct, and use the transfer function when transferring BridgeCap instead of using public_transfer:

```
// bridge.move --> BridgeCap
-struct BridgeCap has key, store {
+struct BridgeCap has key {
    id: UID
}
// bridge.move --> init()
- public_transfer(cap, tx_context::sender(ctx));
+ transfer(cap, tx_context::sender(ctx));
```

# 4. CONCLUSION

In this audit, we thoroughly analyzed **Mango** 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 |
| **Result** | Not found |
| **Severity** | HIGH |

### 5.2.2 Account Permission Control

| | |
|---|---|
| **Description** | Examine permission control issue in the contract |
| **Result** | Not found |
| **Severity** | MEDIUM |

#### 5.2.3 Malicious Code Behavior

| | |
|---|---|
| **Description** | Examine whether sensitive behavior present in the code |
| **Result** | 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 |
| **Result** | 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

# Contact

Website
www.exvul.com

Email
contact@exvul.com

Twitter
@EXVULSEC

Github
github.com/EXVUL-Sec

ExVul