# Vottun Bridge Smart Contracts Secure Code Review

Technical Report

## Qubic

14 August 2025
Version: 2.1

Kudelski Security – Nagravision Sàrl

Corporate Headquarters
Kudelski Security – Nagravision Sàrl
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

For Public Release

# DOCUMENT PROPERTIES

| | |
|---|---|
| Version: | 2.1 |
| File Name: | Kudelski_Security_Qubic_Secure_Code_Review_v2.1.pdf |
| Publication Date: | 14 August 2025 |
| Confidentiality Level: | For Public Release |
| Document Recipients: | Axel Diez |
| Document Status: | Approved |

# TABLE OF CONTENTS

# EXECUTIVE SUMMARY

Qubic (“the Client”) engaged Kudelski Security (“Kudelski”, “we”) to perform the Vottun Bridge Smart Contracts Secure Code Review.

The assessment was conducted remotely by the Kudelski Security Team.

The review took place between 17 July 2025 and 28 July 2025, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

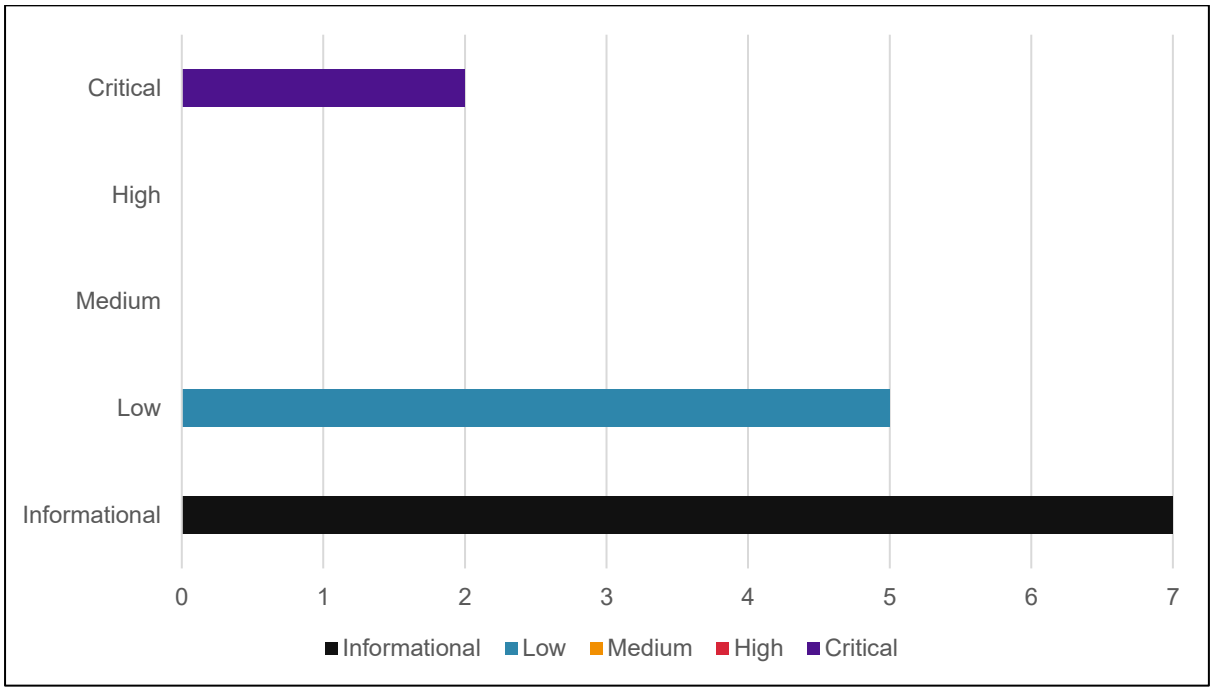
A second review was performed on 14 August 25 on a modified version of the code.

## Key Findings

The following are the major themes and issues identified during the testing period.

These, along with other items within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- Missing Token Lock `createOrder` Function
- Refunding Ethereum-to-Qubic Orders Without Sufficient Locked Tokens Can Cause Underflow, or Denial of Service
- Risks related to role-based and centralization
- Missing Checksum Validation for Qubic Addresses



# 1. PROJECT SUMMARY

This report summarizes the engagement, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

## 1.1 Context

The code reviewed were smart contracts that will be deployed to build a bridge between Qubic token and the Ethereum blockchain.

## 1.2 Scope

The scope consisted in specific Solidity files and folders located at:

- Commit hash: `b60cb701db1a1bb4286505f3e15dd97ed6760073`
- Source code repository: https://github.com/vottundev/vottun-qubic-bridge-sc-evm ,

The files and folders in scope are:

- `QubicBridge.sol`
- `QubicToken.sol`
- `IQubicBridge.sol`
- `IQubicToken.sol`

The scope also included in specific C++ header file and folders located at:

- Commit hash: `cb1724f7b57dfb9e66dc6a0038e5044c966d1fdd`
- Source code repository:

The files and folders in scope are: https://github.com/sergimima/core/blob/testnets/2025-05-12-release245/src/contracts/VottunBridge

- `src/contracts/VottunBridge.h`

After the initial review, Vottun Bridge A rereview was performed on the modified code on 14 August 2025 on the following repositories:

- https://github.com/vottundev/vottun-qubic-bridge-sc-evm/
  - Commit hash: `74a84a733bc57118090043b57e931f501ebf6fe7`
- https://github.com/sergimima/core/blob/testnets/2025-05-12-release-245/src/contracts/VottunBridge.h
  - Commit hash: `0f998c02cb1d274fd4bd5263c46187c2b7c24aee`

## 1.3 Remarks

During the code review, the following positive observations were noted regarding the scope of the engagement:

- The code was well-commented which helps for better understanding of the implementation of the bridge.
- Tests were also provided as part of the project, which is convenient for better understanding how the implementation works and useful for elaborating scenarios and validating findings.

## 1.4 Additional Note

It is important to notice that, although we did our best in our analysis, no code audit assessment is per se guarantee of absence of vulnerabilities. Our effort was constrained by resource and time limits, along with the scope of the agreement.

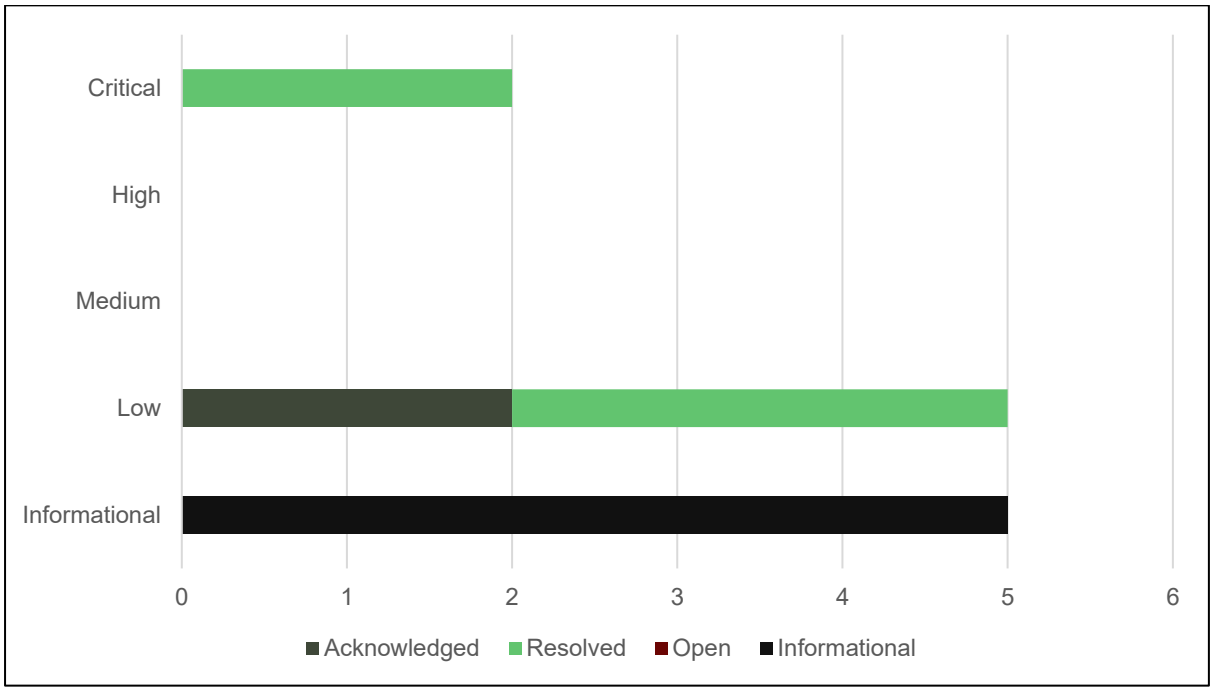
In assessing the severity of some of the findings we identified, we kept in mind both the ease of exploitability and the potential damage caused by an exploit.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination. Information about the severity ratings can be found in Chapter Vulnerability Scoring System of this document.

## Status of Findings after Re-review

The Client modified the source code based on the recommendations in this report. Subsequently, The Kudelski Security Team has evaluated these changes and updated the status of each finding:

- **Resolved** – the Client did modify the implementation, and we considered the fix to be correct.
- **Acknowledged** – the Client decided to accept the risk, or the mitigation is difficult or impossible to implement with the existing tools and resources.
- **Open** – The issue remains unresolved




Overview of findings and their status.

## 2. TECHNICAL DETAILS OF SECURITY FINDINGS

This chapter provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

| # | SEVERITY | TITLE | STATUS |
|---|---|---|---|
| KS–VB–F–01 | Critical | Missing Token Lock `createOrder` Function | Resolved |
| KS–VB–F–02 | Critical | Refunding Ethereum-to-Qubic Orders Without Sufficient Locked Tokens Can Cause Underflow | Resolved |
| KS–VB–F–03 | Low | Risk of Denial of Service due to Fixed-sized `orders` Array | Resolved |
| KS–VB–F–04 | Low | Centralization Risks | Acknowledged |
| KS–VB–F–05 | Low | Floating Pragmas | Resolved |
| KS–VB–F–06 | Low | Lack of Input Sanitization | Resolved |
| KS–VB–F–07 | Low | Missing Checksum Validation | Acknowledged |

Findings overview.

## 2.1 KS–VB–F–01 Missing Token Lock `createOrder` Function

| Severity | Impact | Likelihood | Status |
|---|---|---|---|
| Critical | High | High | Resolved |

### Description

The `createOrder` function in the `VottunBridge.h` contract has a critical security vulnerability where it does not properly lock tokens when creating an order. While the function verifies the user has sufficient balance for fees, it fails to actually transfer and lock the tokens being bridged. This allows users to request refunds for tokens that were never actually locked in the contract.

### Impact

Users can create bridge orders without committing their tokens.

Users can request refunds through `refundOrder` for tokens they never transferred.

Financial loss for the protocol and deterioration of bridge security guarantees.

### Evidence

In `VottunBridge.h`, the `createOrder` function, defined at lines 296-395, does no verification including the variable `state.lockedTokens`. Therefore, users can create Ethereum-to-Qubic orders even though locked tokens are not sufficient.

```
uint64 requiredFeeEth = (input.amount * state._tradeFeeBillionths) /
1000000000ULL;
uint64 requiredFeeQubic = (input.amount * state._tradeFeeBillionths) /
1000000000ULL;
uint64 totalRequiredFee = requiredFeeEth + requiredFeeQubic;

// Verify that the fee paid is sufficient for both fees
if (qpi.invocationReward() < static_cast<sint64>(totalRequiredFee))
{
    // Fee check error handling
    return;
}
```

`VottunBridge.h`. the `createOrder` function only checks fees.

### Affected Resources

- `VottunBridge.h` lines 296-395, 833

### Recommendation

Add validation in `refundOrder` to verify tokens were locked.

Add proper accounting of locked vs received tokens.

## 2.2 KS–VB–F–02 Refunding Ethereum-to-Qubic Orders Without Sufficient Locked Tokens Can Cause Underflow

| Severity | Impact | Likelihood | Status |
|---|---|---|---|
| Critical | High | High | Resolved |

### Description

The contract allows the creation of Ethereum-to-Qubic orders (`fromQubicToEthereum == false`) even when `state.lockedTokens` is zero, which is the case at the contract deployment. If such an order is created, it cannot be completed due to insufficient locked tokens. Furthermore, if a refund is attempted on this order, the contract will execute `state.lockedTokens -= order.amount;` without checking for underflow, which can cause the `lockedTokens` value to wrap around to a very large number or potentially crash the contract, depending on the platform's integer handling.

### Impact

Users may be unable to complete or refund Ethereum-to-Qubic orders if there are no locked tokens, resulting in stuck funds and failed bridge operations.

Attempting to refund such orders can cause an underflow in `state.lockedTokens`, leading to incorrect contract state, potential loss of accounting integrity, and possible exploitation or denial of service.

### Evidence

In `VottunBridge.h`, the `createOrder` function, defined at lines 296-395, does no verification including the variable `state.lockedTokens`. Therefore, users can create Ethereum-to-Qubic orders even though locked tokens are not sufficient.

```
// Ensure sufficient tokens are locked for the order
          if (state.lockedTokens < locals.order.amount)
          {
              locals.log = EthBridgeLogger{
                  CONTRACT_INDEX,
                  EthBridgeError::insufficientLockedTokens,
                  input.orderId,
                  locals.order.amount,
                  0};
              LOG_INFO(locals.log);
              output.status = EthBridgeError::insufficientLockedTokens; // Error
              return;
```

```
            }
```

VottunBridge.h. In the completeOrder procedure, the contract checks if (state.lockedTokens < order.amount) and fails if not enough tokens are locked.

```
// Update the status and refund tokens
        qpi.transfer(locals.order.qubicSender, locals.order.amount);
        state.lockedTokens -= locals.order.amount;
        locals.order.status = 2;                    // Refunded
        state.orders.set(locals.i, locals.order); // Use the loop index instead of
```

VottunBridge.h. In the refundOrder procedure, the contract executes state.lockedTokens -= order.amount; without verifying that lockedTokens is sufficient, risking underflow.

### Affected Resources

- VottunBridge.h lines 296-395, 710, 833

### Recommendation

Consider preventing the creation of Ethereum-to-Qubic orders when there are insufficient locked tokens or provide clear user feedback about the contract's locked token state.

Implement underflow/overflow protection for all arithmetic operations on critical state variables to ensure contract safety and integrity.

## 2.3 KS–VB–F–03 Risk of Denial of Service Due to Fixed-Size `orders` Array

| Severity | Impact | Likelihood | Status |
|---|---|---|---|
| Low | Low | Low | Resolved |

### Description

Once all 1024 slots are filled with active orders, no new orders can be created until an existing slot is freed (e.g., by refunding or completing and marking as empty). If the array is full, any attempt to create a new order will fail, resulting in a denial of service (DoS) for all users.

### Impact

Attackers or heavy usage could fill the array, blocking legitimate users from using the bridge. Then, the bridge becomes unusable until slots are manually freed, which may not be possible if orders are stuck or intentionally left incomplete.

As an attacker needs to pay fees for each order created, this is considered as a finding with a low severity.

### Evidence

```
Array<BridgeOrder, 1024> orders;
```

`VottunBridge.h`. Array `orders` used to store all orders perform on the bridge.

### Affected Resources

- `VottunBridge.h`

### Recommendation

Consider using a dynamic data structure or implement a mechanism to recycle or prune old/completed orders.

Monitor array usage and alert if approaching capacity.

Consider increasing the array size or using a mapping if platform constraints allow.

## 2.4 KS–VB–F–04 Centralization Risks

| Severity | Impact | Likelihood | Status |
|---|---|---|---|
| Low | Low | Low | Acknowledged |

### Description

The Vottun bridge implements different roles for the good functioning of the bridge and exhibits significant centralization risks due to its privileged role structure:

- Operator Role Power: Operators have the authority to confirm, revert, and execute orders. They can arbitrarily set the `feeRecipient` address, allowing them to direct transfer fees to themselves or any address they control. This creates a risk of fee extraction and self-dealing, especially if operator identities are not transparent or subject to off-chain governance.
- Admin Role Power: The admin can withdraw all tokens and Ether from the contract at any time using emergency withdrawal functions. If the admin's private key is compromised or the admin acts maliciously, all user funds in the bridge can be taken.

There are no on-chain mechanisms to prevent operators or the admin from abusing their privileges. The contracts rely entirely on off-chain trust and governance.

### Impact

Users must trust that operators and the admin will act honestly and not abuse their powers. This centralization introduces custodial risk, and the contract is not trustless.

### Evidence

```
    /**
     * @notice Called by the admin to withdraw tokens in case of emergency
     * @param tokenAddress Address of the token to withdraw
     * @param amount Amount of tokens to withdraw
     */
    function emergencyTokenWithdraw(address tokenAddress, uint256 amount) external
onlyRole(DEFAULT_ADMIN_ROLE) nonReentrant {
        (bool success, ) =
tokenAddress.call(abi.encodeWithSignature("transfer(address,uint256)", msg.sender,
amount));

        if (!success) {
            revert TokenTransferFailed();
        }
```

```
        emit EmergencyTokenWithdrawn(tokenAddress, msg.sender, amount);
    }

    /**
     * @notice Called by the admin to withdraw all Ether in case of emergency
     */
    function emergencyEtherWithdraw() external onlyRole(DEFAULT_ADMIN_ROLE)
nonReentrant {
        uint256 amount = address(this).balance;
        (bool success, ) = msg.sender.call{value: amount}("");

        if (!success) {
            revert EtherTransferFailed();
        }

        emit EmergencyEtherWithdrawn(msg.sender, amount);
    }
```

`QubicBridge.sol`. Emergency function that allows the admin to withdraw all tokens

### Affected Resources

- `QubricBrige.sol`
- `VottunBridge.h`

### Recommendation

We understand the need for role-based access and emergency withdrawal functions but believe this should be transparent to bridge users. We also recommend using multi-signature accounts for high-responsibility roles like administrators.

### References

https://arxiv.org/abs/2312.06510

## 2.5 KS–VB–F–05 Floating Pragmas

| Severity | Impact | Likelihood | Status |
|---|---|---|---|
| Low | Low | Low | Resolved |

### Description

It is best practice to deploy smart contracts that have been built with the exact same version of the compiler that have been used for testing.

Furthermore, vulnerabilities exist in some compiler versions above 0.8.0.

### Impact

Floating pragmas will allow any Solidity compiler of a higher versions to be used to compile the code. This also includes nightly builds of the Solidity compiler.

If an unstable compiler is used to compile the code to be released, the deployed smart contract may be unstable and in worst case buggy or vulnerable.

### Evidence

```
pragma solidity ^0.8.28
```

`QubicBridge.h`. The smart contract can be compiled with solidity 0.8.28 or later.

### Affected Resources

- `QubicBridge.sol`
- `QubicToken.sol`
- `IQubicBridge.sol`
- `IQubicToken.sol`

### Recommendation

Do not use floating pragmas. Use the same version of the compiler to both test and deploy the smart contracts, ideally the latest, stable one.

### References

- https://swcregistry.io/docs/SWC-103
- Solidity: Strict pragma vs Floating pragma

## 2.6 KS–VB–F–06 Lack of Input Sanitization

| Severity | Impact | Likelihood | Status |
|---|---|---|---|
| Low | Low | Low | Resolved |

### Description

The functions `confirmOrder` and `revertOrder` accept a `feePct` parameter, representing the percentage of the transfer fee. However, there is no input validation to ensure that `feePct` does not exceed 100%. This allows an operator to set `feePct` to any value, including values greater than 100%, which could result in the fee exceeding the transferred amount.

### Impact

These could be the possible impacts of the incorrect inputs:

- Excessive Fee Extraction: If `feePct` is set above 100%, the calculated fee can exceed the order amount, potentially draining user funds or causing unexpected behaviour.
- Loss of Funds: The recipient may receive less than expected, or the protocol may transfer or burn more tokens than intended.
- Protocol Abuse: Malicious or misconfigured operators could exploit this to extract excessive fees or disrupt the bridge's operation.

### Evidence

```
function revertOrder(
        uint256 orderId,
        uint256 feePct,
        address feeRecipient
    ) external onlyRole(OPERATOR_ROLE) nonReentrant {
        PullOrder memory order = pullOrders[orderId];
        uint256 amount = uint256(order.amount);

        if (amount == 0) {
            revert InvalidOrderId();
        }
        if (order.done) {
            revert AlreadyConfirmed();
        }
        if (feeRecipient == address(0)) {
```

```
            revert InvalidFeeRecipient();
        }
```

`QubicBridge.sol`. Example with the revertOrder function of the lack of input sanitization

### Affected Resources

- `VottunBridge.h`

### Recommendation

Add an input validation check to ensure `feePct` does not exceed 100% in both `confirmOrder` and `revertOrder`, like the check already present in `executeOrder`.

## 2.7 KS–VB–F–07 Missing Checksum Validation

| Severity | Impact | Likelihood | Status |
|---|---|---|---|
| Low | Low | Low | Acknowledged |

### Description

Based on [1] the public key on Qubic is encoded into an alphanumeric string, the last part of it representing a checksum. In the code, the function `isQubicAddress`:

- Checks (redundantly) if the characters are from 0-9;
- Does not check if the checksum is correct.

### Impact

Not checking if an address is valid might lead to incorrect transactions or loss of funds.

### Evidence

```
/**
 * @notice Checks if an address is a valid Qubic address
 * @param addr Address to check
 * @return bool
 */
function isQubicAddress(string memory addr) internal pure returns (bool) {
    bytes memory baddr = bytes(addr);

    if (baddr.length != QUBIC_ACCOUNT_LENGTH) {
        return false;
    }

    for (uint i = 0; i < QUBIC_ACCOUNT_LENGTH; i++) {
        bytes1 char = baddr[i];

        if (
            !(char >= 0x30 && char <= 0x39) && // 0-9
            !(char >= 0x41 && char <= 0x5A) // A-Z
        ) {
            return false;
        }
    }
```

```
        return true;
    }
```

`QubicBridge.sol` lines 424-449

### Affected Resources

- `QubicBridge.sol` lines 424-449

### Recommendation

Correct the code to perform the checksum verification as well.

### References

[1] https://github.com/qubic/qubic-cli/blob/main/keyUtils.cpp

# 3. OBSERVATIONS

This chapter contains additional observations that are not directly related to the security of the code, and as such have no severity rating or remediation status summary. These observations are either minor remarks regarding good practice or design choices or related to implementation and performance. These items do not need to be remediated for what concerns security, but where applicable we include recommendations.

| # | SEVERITY | TITLE | STATUS |
|---|---|---|---|
| KS–VB–O–01 | Informational | Dead Code and Duplicated Code | Informational |
| KS–VB–O–02 | Informational | Incorrect Message Error | Informational |
| KS–VB–O–03 | Informational | Dependency Added using Unstable Branch | Informational |
| KS–VB–O–04 | Informational | Debugging Code | Informational |
| KS–VB–O–05 | Informational | Zero Address Verification Not Performed | Informational |
| KS–VB–O–06 | Informational | Ethereum Addresses Storage Inconsistencies | Informational |
| KS–VB–O–07 | Informational | `amount` Verification Inconsistencies | Informational |

Observations overview.

## 3.1 KS–VB–O–01 Dead Code and Duplicate Code

### Description

The contract defines an internal function `isAdmin`, which checks if the caller is the current admin. However, this function is never invoked anywhere in the contract. All admin checks are performed directly by comparing qpi.`invocator`() to `state.admin` within the relevant procedures (e.g., `setAdmin`, `addManager`, `removeManager`, `withdrawFees`).

Additionally, the declared structure `VOTTUNBRIDGE2` is never used.

### Affected Resources

- `VottunBridge.h`

### Recommendation

Refactor the contract to use `isAdmin` for all admin checks, ensuring consistent and maintainable access control logic. Remove all unused code.

## 3.2 KS–VB–O–02 Incorrect Message Error

### Description

The error code `onlyManagersCanCompleteOrders` is used not only when managers attempt to complete orders, but also when they attempt to revert (refund) orders. This can cause confusion, as the same error code is used for two distinct actions: completing and reverting orders.

### Affected Resources

### Recommendation

Rename the error code to a more general and accurate name, such as `onlyManagersCanRevertOrders` or `onlyManagersCanManageOrders`, or introduce a separate error code for revert/refund actions. This will improve clarity and make error handling more precise.

## 3.3 KS–VB–O–03 Dependency Added using Unstable Branch

### Description

The `README.md` of the OpenZeppelin project has the following warning:

> *When installing via git, it is a common error to use the master branch. This is a development branch that should be avoided in favor of tagged releases. The release process involves security measures that the master branch does not guarantee.*

### Affected Resources

The `.gitmodules` file.

### Recommendation

When adding the OpenZeppelin project as a dependency, use a release branch and not the master branch (which is explicitly marked as development-only and potentially unstable).

## 3.4 KS–VB–O–04 Debugging Code

### Description

Code used for debugging is still present in the code.

### Affected Resources

- `VottunBridge.h` lines 185, 994-10011100-43

### Recommendation

Remove the debugging code before going to production.

## 3.5 KS–VB–O–05 Zero Address Verification not Performed

### Description

In the contract `QubicToken.sol`, the functions `setAdmin` and `addOperator` do not validate that the provided address (`newAdmin` or `newOperator`) is not the zero. Assigning roles to the zero address can lead to loss of control over the contract or inability to manage operators/admins.

### Affected Resources

- `QubicToken.h` lines 24-43

### Recommendation

Add a check to ensure that `newAdmin` and `newOperator` are not the zero address before granting roles as performed in the `QubicBridge.sol` contract.

## 3.6 KS–VB–O–06 Ethereum Addresses Storage Inconsistencies

### Description

The contract defines the field `ethAddress` as `Array<uint8, 64>` to store Ethereum addresses, but only the first 42 elements are used for actual address data in the functions `createOrder`, `getOrderByDetails`. The remaining 22 elements are unused.

### Affected Resources

- `VottunBridge.h` lines 351-354,1053-1060

### Recommendation

Add a check to ensure that `newAdmin` and `newOperator` are not the zero address before granting roles as performed in the `QubicBridge.sol` contract.

## 3.7 KS–VB–O–07 `amount` Verification Inconsistencies

### Description

In `VottunBridge.h`, the `createOrder` procedure only checks that the input `amount` is not zero, but does not enforce any upper limit on the amount that can be transferred in a single order as it is performed on the solidity implementation `QuBridge.sol`.

### Affected Resources

- `VottunBridge.h` line 299

### Recommendation

Follow the same logic for both smart contracts.

# 4. METHODOLOGY

For this engagement, Kudelski Security used a methodology that is described at a high level in this chapter. This is broken up into the following phases.


## 4.1 Kickoff

The Kudelski Security Team set up a kickoff meeting where project stakeholders were gathered to discuss the project as well as the responsibilities of participants. During this meeting, we verified the scope of the engagement and discussed the project activities.

## 4.2 Ramp-up

Ramp-up consisted of the activities necessary to gain proficiency on the particular project. This included the steps required for gaining familiarity with the codebase and technological innovations utilized.

## 4.3 Review

The review phase is where most of the work on the engagement was performed. In this phase we have analyzed the project for flaws and issues that could impact the security posture. The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools was used to assist the reviewer during the testing. We discuss our methodology in more detail in the following subsections.

### Code Review

Kudelski Security Team reviewed the code within the project utilizing an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content, to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories:

- authentication (*e.g.* A07:2021, CWE-306)
- authorization and access control (*e.g.* A01:2021, CWE-862)
- auditing and logging (*e.g.* A09:2021)
- injection and tampering (*e.g.* A03:2021, CWE-20)
- configuration issues (*e.g.* A05:2021, CWE-798)
- logic flaws (*e.g.* A04:2021, CWE-190)
- cryptography (*e.g.* A02:2021)

These categories incorporate common weaknesses and vulnerabilities such as the OWASP Top 10 and MITRE Top 25.

### Smart Contracts

We reviewed the smart contracts, checking for additional specific issues that can arise such as:

- assessment of smart contract admin centralization
- reentrancy attacks and external contracts interactions
- verification of compliance with existing standards such as ERC20 or PSP34
- unsafe arithmetic operations such as overflow and underflow
- verification dependance on timestamp
- access control verification to ensure that only authorized users can call sensitive functions.

## 4.4 Reporting

Kudelski Security delivered to the Client a preliminary report in PDF format that contained an executive summary, technical details, and observations about the project.

In the report we not only point out security issues identified but also observations for improvement. The findings are categorized into several buckets, according to their overall severity: **Critical**, **High**, **Medium**, **Low**.

Observations are considered to be **Informational**. Observations can also consist of code review, issues identified during the code review that are not security related, but are general best practices and steps, that can be taken to lower the attack surface of the project.

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

## 4.5 Verify

After the preliminary findings have been delivered, we verify the fixes applied by the Client. After these fixes were verified, we updated the status of the finding in the report.

The output of this phase is the final report with any mitigated findings noted.

# 5. VULNERABILITY SCORING SYSTEM

Kudelski Security utilizes a custom approach when computing the vulnerability score, based primarily on the **Impact** of the vulnerability and **Likelihood** of an attack.

Each metric is assigned a ranking of either low, medium or high, based on the criteria defined below. The overall severity score is then computed as described in the next section.

## Severity

Severity is the overall score of the finding, weakness or vulnerability as computed from Impact and Likelihood. Other factors, such as availability of tools and exploits, number of instances of the vulnerability and ease of exploitation might also be taken into account when computing the final severity score.

| IMPACT / LIKELIHOOD | LOW | MEDIUM | HIGH |
|---|---|---|---|
| **HIGH** | Medium | High | High |
| **MEDIUM** | Low | Medium | High |
| **LOW** | Low | Low | Medium |

Compute overall severity from Impact and Likelihood. The final severity factor might vary depending on a project's specific context and risk factors.

- **Critical** The identified issue may be immediately exploitable, causing a strong and major negative impact system-wide. They should be urgently remediated or mitigated.
- **High** The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
- **Medium** The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
- **Low** The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
- **Informational** findings are best practice steps that can be used to harden the application and improve processes. Informational findings are not assigned a severity score and are classified as Informational instead.

## Impact

The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

- **High** The vulnerability has a severe effect on the company and systems or has an effect within one of the primary areas of concern noted by the client.
- **Medium** It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.
- **Low** There is little to no effect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

## Likelihood

The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, difficulty of exploitation and institutional knowledge.

- **High** It is extremely likely that this vulnerability will be discovered and abused.
- **Medium** It is likely that this vulnerability will be discovered and abused by a skilled attacker.
- **Low** It is unlikely that this vulnerability will be discovered or abused when discovered.

# 6. CONCLUSION

The objective of this Secure Code Review was to evaluate whether there were any vulnerabilities that would put Qubic or its customers at risk.

The Kudelski Security Team identified 7 security issues: 2 findings are critical, and 5 findings are lower risks. On average, the effort needed to mitigate these risks is estimated as medium to low.

In order to mitigate the risks posed by this engagement's findings, the Kudelski Security Team recommends applying the following best practices:

- Refunding of Ethereum-to-Qubic orders could break the smart contract logic
- Ensuring that tokens were locked before refunding.
- Risk related to centralization
- Improve input sanitization
- Floating Pragmas
- Missing Checksum Validation

Kudelski Security remains at your disposal should you have any questions or need further assistance.

Kudelski Security would like to thank Qubic for their trust, help and support over the course of this engagement and is looking forward to cooperating in the future.

## DOCUMENT RECIPIENTS

| NAME | POSITION | CONTACT INFORMATION |
|---|---|---|
| Axel Diez | Project Manager | axel@qubic.org |

## KUDELSKI SECURITY CONTACTS

| NAME | POSITION | CONTACT INFORMATION |
|---|---|---|
| Jean-Sebastien Nahon | Application and Blockchain Security Practice Manager | jean-sebastien.nahon@kudelskisecurity.com |
| Ana Acero | Project Manager/ Operations Coordinator | ana.acero@kudelskisecurity.com |

## DOCUMENT HISTORY

| VERSION | DATE | STATUS/ COMMENTS |
|---|---|---|
| 1.0 | 28 July 2025 | First Draft |
| 2.0 | 14 August 2025 | Proposal |
| 2.1 | 14 August 2025 | Approved |