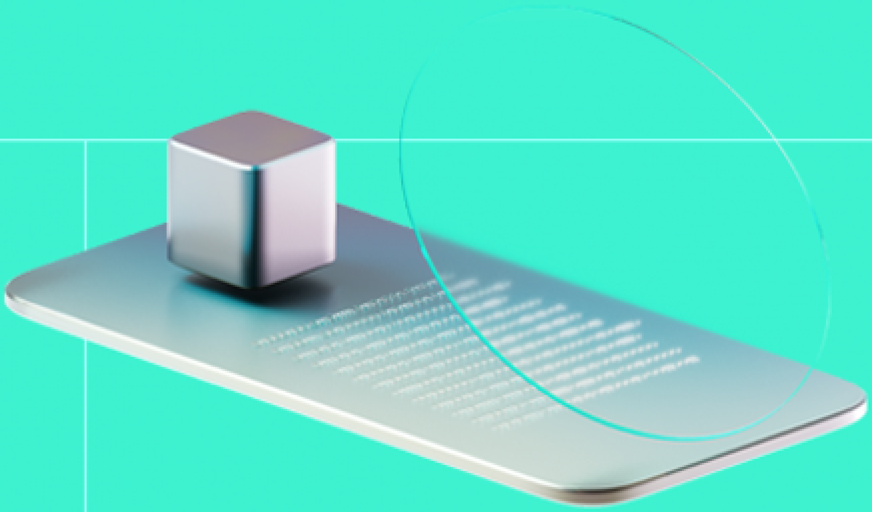
Smart Contract Code Review And Security Analysis Report



Customer: Bitlayer

Date: 08/04/2024

We express our gratitude to the Bitlayer team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Bitlayer represents a revolutionary integration, melding elite decentralized exchange (DEX) mechanisms into an innovative, high-efficiency system. This pioneering approach is designed to streamline token exchanges and cross-chain transactions, ensuring seamless operability and liquidity management within the crypto ecosystem.

Platform: EVM

Language: Solidity

Tags: Bridge

Timeline: 02/04/2024 - 04/04/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/bitlayer-org/bitlayer-bridge
Commit	41c7c06
Repository	https://github.com/bitlayer-org/getBTC
Commit	345036b

Audit Summary

10/10	9/10	86.36%	7/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9/10

The system users should acknowledge all the risks summed up in the risks section of the report

5	3	1	1
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	1
Low	3

Vulnerability	Status
F-2024-1925 - Missing return value check on permit function	Mitigated
F-2024-1957 - Missing two-step ownership transfer process	Accepted
F-2024-1915 - Missing checks for the zero address	Fixed
F-2024-1926 - Missing array length cache in for loop	Fixed
F-2024-1956 - Missing lock limitations	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Bitlayer
Audited By	Kaan Caglan
Approved By	Przemyslaw Swiatowiec
Website	https://www.bitlayer.org/
Changelog	04/04/2024 - Preliminary Report
	08/04/2024 - Final Report

Table of Contents

System Overview **6**

Executive Summary **8**

Documentation Quality 8

Code Quality 8

Test Coverage 8

Security Score 8

Summary 8

Risks **9**

Findings **10**

Vulnerability Details 10

Observation Details 18

Disclaimers 25

Appendix 1. Severity Definitions **26**

Appendix 2. Scope **27**

System Overview

Bitlayer bridge comprises three main contracts:

TokenExchange: Facilitates the exchange of tokens with rigorous permission and signature verification mechanisms. It supports operations like swapping tokens under specific conditions, withdrawing tokens, and modifying contract administrative roles.

BitlayerProxy: Serves as a proxy following the ERC1967 standard, allowing for future upgrades and changes to the contract logic without affecting the deployed version.

BitlayerBridge: Enables the locking and unlocking of native cryptocurrency transactions across different blockchain networks, with roles and permissions managed through AccessControl. It supports liquidity management, fee adjustments, and pausable functionality for emergency stops.

Contracts

TokenExchange

- **Functionalities**:
  - Facilitates token swaps with ERC20 tokens using EIP712 signatures for permissioned operations.
  - Allows the withdrawal of ERC20 tokens and native cryptocurrency (referred to as "BTC" in comments) by the owner.
  - Supports owner and operator role management for executing sensitive contract operations.
  - Manages vaults that designate supported token addresses for swapping.
- **Attributes**:
  - `owner`: Address of the contract owner.
  - `operator`: Address of the contract operator.
  - `vaults`: A mapping of addresses to boolean values, indicating whether a token address is supported for swaps.
- **Privileged Roles**:
  - `owner`: Can transfer ownership, set the operator, manage vaults, and withdraw tokens.
  - `operator`: Can execute the `permitAndSwap` function.

BitlayerProxy

- **Functionalities**: Serves as a minimalistic instance of the ERC1967Proxy, primarily for deployment purposes without additional specific functionalities.

BitlayerBridge

- **Functionalities**:
  - Manages cross-chain transactions with functionality to lock and unlock native cryptocurrency.
  - Handles liquidity contributions and withdrawals, enabling users to support the bridge's operations with their assets.
  - Allows role-based management for pausing and unpausing contract operations, setting fee addresses, and adjusting fees.
- **Attributes**:

  - `feeAddress`: Address where transaction fees are collected.
  - `lockFeeAmount`: The amount charged as a fee for locking transactions.
  - `liquidityOf`: Mapping of addresses to their contributed liquidity amounts.
  - `totalLocked`: Total amount of cryptocurrency locked through the bridge.
  - `totalUnlocked`: Total amount of cryptocurrency unlocked.
- **Privileged Roles**:
  - `AdminRole`: Can upgrade the contract, manage roles, pause/unpause the bridge, and adjust fees and the fee address.
  - `PauseRole`: Can pause contract operations.
  - `UnlockRole`: Can unlock transactions, allowing the withdrawal of locked funds.
  - `LiquidityRole`: Can manage liquidity withdrawals on behalf of others.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **7** out of **10**.

- Functional requirements are partially missed.
- Technical description is not provided.
- Technical flow is not provided.

Code quality

The total Code Quality score is **9** out of **10**.

- Missing best practices
- The development environment is configured.

Test coverage

Code coverage of the project is **86.36%** (branch coverage),

- Not all branches are covered with tests.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **1** medium, and **3** low severity issues. After remediation part of the audit process **1** medium issue was mitigated, **2** low issues were fixed and 1 low issue was accepted, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.** This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- Authorized used can call `unlock` function anytime to get balances in the contract..
- Solidity version `0.8.20` might not work on all chains due to `PUSH0` opcode.
- There is no upper limit set for `lockFeeAmount`, allowing the admin to assign any valid integer value to it.

Findings

Vulnerability Details

F-2024-1925 - Missing return value check on permit function - Medium

Description: The Anyswap hack occurred because the `permit()` function didn't really exist, but the fallback function that took its place did not complain. Consider using `safePermit()` which ensures that the permit actually went through.

```
IERC20Permit(tokenAddress).permit(approver, address(this), amountIn,
deadline, pv, pr, ps);
```

If a token that lacks a `permit` function and has a non-reverting `fallback` function is passed to the `permitAndSwap` function, the function will not revert, even if the signature provided is incorrect.

Assets:

- contracts/TokenExchange.sol [https://github.com/bitlayer-org/getBTC]

Status: Mitigated

Classification

Severity: Medium

Impact: Likelihood [1-5]: 2
Impact [1-5]: 5
Exploitability [0-2]: 1
Complexity [0-2]: 0
Final Score: 2.7 (Medium)
Hacken Calculator Version: 0.6

Recommendations

Recommendation: Consider replacing the use of the `permit()` function with `safePermit()` from OpenZeppelin's SafeERC20 library or a similar safe implementation. `safePermit()` provides additional safety checks to ensure that the permit transaction is executed securely, reducing the risk of potential vulnerabilities.

Remediation (Revised commit: 8248293): The Bitlayer team accepted that they are aware of the risk and they will only use tokens that they are sure.

F-2024-1915 - Missing checks for the zero address - Low

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "zero address". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address. The "Missing zero address control" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.

For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

```
feeAddress = _feeAddress;
operator = newOp;
```

Assets:

- contracts/TokenExchange.sol [https://github.com/bitlayer-org/getBTC]
- contracts/BitlayerBridge.sol [https://github.com/bitlayer-org/bitlayer-bridge]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 1
Impact [1-5]: 5
Exploitability [0-2]: 2
Complexity [0-2]: 0
Final Score: 1.8 (Low)
Hacken Calculator Version: 0.6

Recommendations

Recommendation:	It is strongly recommended to implement checks to prevent the zero address from being set during the initialization of contracts. This can be achieved by adding require statements that ensure address parameters are not the zero address. **Remediation (Revised commit: 8248293):** The Bitlayer team fixed the issue by adding zero checks.

F-2024-1956 - Missing lock limitations - Low

Description:

Lock and unlock amounts are currently unrestricted, with a minimum value requirement for locking that applies only to fees and no limitations on either action. For safety purposes, an upper limit for locks should be introduced.

```
function lock(string memory to) external whenNotPaused payable {
require(msg.value > lockFeeAmount, "not enough fee");

(bool success, bytes memory returndata) = feeAddress.call{value: loc
kFeeAmount}("");
require(success, string(returndata));

uint256 lockedAmount = msg.value - lockFeeAmount;
totalLocked += lockedAmount;

emit NativeLocked(msg.sender, to, lockedAmount, lockFeeAmount);
}

function unlock(string memory _txHash, address payable to, uint256 a
mount)
external
onlyRole(UnlockRole)
whenNotPaused
{
bytes32 txHash = keccak256(abi.encode(_txHash));
require(!txUnlocked[txHash], "txHash already unlocked");
txUnlocked[txHash] = true;

(bool success, bytes memory returndata) = payable(to).call{value: am
ount}("");
require(success, string(returndata));

totalUnlocked += amount;

emit NativeUnlocked(_txHash, to, amount);
}
```

Assets:

- contracts/BitlayerBridge.sol [https://github.com/bitlayer-org/bitlayer-bridge]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 1
Impact [1-5]: 5
Exploitability [0-2]: 2
Complexity [0-2]: 0
Final Score: 1.8 (Low)
Hacken Calculator Version: 0.6

Recommendations

Recommendation: A state variable should be implemented, modifiable by an administrator, to facilitate comparison between locked and unlocked amounts.

Remediation (Revised commit: 1c4f70e): The Bitlayer team fixed the issue by adding min-max lock amount controls.

F-2024-1957 - Missing two-step ownership transfer process - Low

Description:

Ownable2Step and Ownable2StepUpgradeable prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

```
function transferOwnership(address newOwner) external onlyOwner {
require(newOwner != address(0),"Owner_Should_Not_Zero_Address");
owner = newOwner;
emit TransferOwnership(newOwner);
}
```

Assets:

- contracts/TokenExchange.sol [https://github.com/bitlayer-org/getBTC]

Status: Accepted

Classification

Severity: Low

Impact:

Likelihood [1-5]: 1
Impact [1-5]: 5
Exploitability [0-2]: 2
Complexity [0-2]: 0
Final Score: 1.8 (Low)
Hacken Calculator Version: 0.6

Recommendations

Recommendation:

Consider using `Ownable2Step` or `Ownable2StepUpgradeable` from OpenZeppelin Contracts to enhance the security of your contract ownership management. These contracts prevent the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call. This two-step ownership transfer process adds an additional layer of security to your contract's ownership management.

Remediation (Revised commit: 752b04b): The Bitlayer team accepted the issue.

F-2024-1926 - Missing array length cache in for loop - Info

Description: Failing to cache the array length when iterating through arrays in Solidity can have significant performance and gas cost implications. In Solidity, array lengths can change during execution due to external calls or storage modifications. When the array length is not cached before entering a loop, it is recomputed with each iteration, leading to unnecessary gas consumption.

```
for (uint256 i = 0; i < pausers.length; ) {
for (uint256 i = 0; i < unlockers.length; ) {
for (uint256 i = 0; i < liquiditiers.length; ) {
```

Assets:

- contracts/BitlayerBridge.sol [https://github.com/bitlayer-org/bitlayer-bridge]

Status: Fixed

Classification

Severity: Info

Recommendations

Recommendation: To enhance performance and reduce gas costs, cache the array length before entering a for loop in Solidity. This approach prevents repeated computation of the array length and mitigates the risk of reentrancy attacks due to array length changes during loop execution.

Remediation (Revised commit: 5b3eb75): The Bitlayer team fixed the issue by caching the array lengths.

Observation Details

F-2024-1918 - TODO comments left in the code - Info

Description: TODO comments are mark areas of code that need attention or completion. These comments serve as reminders for unfinished tasks and can be helpful during the development phase. However, if left untouched in production code, these TODO statements can introduce security vulnerabilities and impact the overall security of a smart contract.

```
// TODO max btc outAomunt is 0.05 BTC
```

Assets:

- contracts/TokenExchange.sol [https://github.com/bitlayer-org/getBTC]

Status: Accepted

Recommendations

Recommendation: It is important to remove TODO comments from production code to avoid potential security vulnerabilities. These comments should be addressed and resolved during the development phase.

Remediation (Revised commit: 752b04b): The Bitlayer team accepted the issue.

F-2024-1919 - Floating Pragma - Info

Description:

The project uses floating pragmas `^0.8.0` and `^0.8.23`.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Assets:

- contracts/TokenExchange.sol [https://github.com/bitlayer-org/getBTC]
- contracts/BitlayerBridge.sol [https://github.com/bitlayer-org/bitlayer-bridge]
- contracts/BitlayerProxy.sol [https://github.com/bitlayer-org/bitlayer-bridge]

Status:

Accepted

Recommendations

Recommendation:

Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs for the compiler version that is chosen.

Remediation (Revised commit: 752b04b): The Bitlayer team accepted the issue.

F-2024-1924 - Typos in the code - Info

Description:

Any typos encountered in the provided documentation/code should be addressed.

```
function setVaults(address valut, bool status) external onlyOwner {
// `vault` instead of `valut`
function verifySignture( // `Signature` instead of `Signture`
```

Assets:

- contracts/TokenExchange.sol [https://github.com/bitlayer-org/getBTC]

Status: Accepted

Recommendations

Recommendation:

Fix typos.

Remediation (Revised commit: 752b04b): The Bitlayer team accepted the issue.

F-2024-1927 - Avoid using state variables directly in `emit` for Gas efficiency - Info

Description:

In Solidity, emitting events is a common way to log contract activity and changes, especially for off-chain monitoring and interfacing. However, using state variables directly in `emit` statements can lead to increased gas costs. Each access to a state variable incurs gas due to storage reading operations. When these variables are used directly in `emit` statements, especially within functions that perform multiple operations, the cumulative gas cost can become significant. Instead, caching state variables in memory and using these local copies in `emit` statements can optimize gas usage.

```
emit NativeLocked(msg.sender, to, lockedAmount, lockFeeAmount); // @
audit-issue: `lockFeeAmount` is a state variable and used on line(s)
: ['139', '142', '137']
```

`lockFeeAmount` can be cached since it is being called more than one time.

Assets:

- contracts/BitlayerBridge.sol [https://github.com/bitlayer-org/bitlayer-bridge]

Status: Fixed

Recommendations

Recommendation:

To optimize gas efficiency, cache state variables in memory when they are used multiple times within a function, including in `emit` statements.

Remediation (Revised commit: 065a0a3): The Bitlayer team fixed the issue by caching the state variable.

F-2024-1929 - Unneeded initializations of uint256 and bool variable to 0/false - Info

Description: In Solidity, it is common practice to initialize variables with default values when declaring them. However, initializing `uint256` variables to `0` and `bool` variables to `false` when they are not subsequently used in the code can lead to unnecessary gas consumption and code clutter. This issue points out instances where such initializations are present but serve no functional purpose.

```
for (uint256 i = 0; i < pausers.length; ) {
for (uint256 i = 0; i < unlockers.length; ) {
for (uint256 i = 0; i < liquiditiers.length; ) {
```

Assets:

- contracts/BitlayerBridge.sol [https://github.com/bitlayer-org/bitlayer-bridge]

Status: Fixed

Recommendations

Recommendation: It is recommended not to initialize integer variables to `0` to and boolean variables to `false` to save some Gas.

Remediation (Revised commit: 5b3eb75): The Bitlayer team fixed the issue by adding zero checks.

F-2024-1930 - Possible Gas optimization by using unchecked subtractions - Info

Description: The `unchecked {}` keyword can be added for subtractions where the operands cannot underflow because of a previous `require()` or `if`-statement. Example scenario:
`require(a <= b); x = b - a` ⇒ `require(a <= b); unchecked { x = b - a }`

```
require(liquidityOf[to] >= amount, "liquidity not enough");
liquidityOf[to] -= amount;
```

```
require(msg.value > lockFeeAmount, "not enough fee");

(bool success, bytes memory returndata) = feeAddress.call{value: lockFeeAmount}("");
require(success, string(returndata));

uint256 lockedAmount = msg.value - lockFeeAmount;
```

Assets:

- contracts/BitlayerBridge.sol [https://github.com/bitlayer-org/bitlayer-bridge]

Status: Fixed

Recommendations

Recommendation: In scenarios where subtraction cannot result in underflow due to prior `require()` or `if`-statements, wrap these operations in an `unchecked` block to save gas. This optimization should only be applied when the safety of the operation is assured. Carefully analyze each case to confirm that underflow is impossible before implementing `unchecked` blocks, as incorrect usage can lead to vulnerabilities in the contract.

Remediation (Revised commit: 1f2b25e): The Bitlayer team fixed the issue by adding unchecked keywords.

F-2024-1984 - Unnecessary payable usage - Info

Description:

The use of the `payable` keyword to convert addresses before sending Ether in Solidity can sometimes be redundant, particularly when the target address (`to`) could be defined as `payable` in the function parameters. This redundancy not only adds unnecessary complexity to the code but also obscures the function's intention of transferring Ether to a specific address. Defining the address as `payable` from the outset clarifies that the function is intended to perform Ether transfers and ensures that the address type is correctly specified for such transactions.

```
function unlock(string memory _txHash, address to, uint256 amount)
external
onlyRole(UnlockRole)
whenNotPaused
{
bytes32 txHash = keccak256(abi.encode(_txHash));
require(!txUnlocked[txHash], "txHash already unlocked");
txUnlocked[txHash] = true;

(bool success, bytes memory returndata) = payable(to).call{value: amount}("");
```

Assets:

- contracts/BitlayerBridge.sol [https://github.com/bitlayer-org/bitlayer-bridge]

Status: Fixed

Recommendations

Recommendation:

Review your contract's functions to identify instances where the `payable` keyword is used to convert addresses just before making a call to transfer Ether. Refactor these functions by specifying the `payable` keyword in the function parameters for addresses intended to receive Ether. This practice enhances code clarity, reduces unnecessary conversions, and explicitly indicates which addresses are expected to participate in Ether transactions.

Remediation (Revised commit: 7b72e52): The Bitlayer team fixed the issue by removing unnecessary keyword.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

[hknio/severity-formula]

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/bitlayer-org/bitlayer-bridge
Commit	41c7c064117218eef147f4ae7e70527088462 73d
Remediation	1f2b25eb3d7f8afef937bdd57be188fbe063abb3
Repository	https://github.com/bitlayer-org/getBTC
Commit	345036b3d5ce868347e5c46ab8a6fe2a071d78df
Remediation	752b04b9b2856cc5f187c2190a07b0a26cf0cead
Whitepaper	Not provided
Requirements	Not provided
Technical Requirements	Not provided

Contracts in Scope

./contracts/TokenExchange.sol

./contracts/BitlayerProxy.sol

./contracts/BitlayerBridge.sol