BridgeToken

Smart Contract Audit Report
Prepared for FishingTown

Date Issued: Dec 22, 2021
Project ID: AUDIT2021048
Version: v1.0
Confidentiality Level: Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Public

Report Information

Project ID	AUDIT2021048
Version	v1.0
Client	FishingTown
Project	BridgeToken
Auditor(s)	Suvicha Buakhom Peeraphut Punsuwan
Author	Peeraphut Punsuwan
Reviewer	Weerawat Pawanawiwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Dec 22, 2021	Full report	Peeraphut Punsuwan

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by FishingTown, Inspex team conducted an audit to verify the security posture of the BridgeToken smart contracts between Nov 30, 2021 and Dec 1, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of BridgeToken smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 1 critical and 2 high-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved or mitigated in the reassessment. Therefore, Inspex trusts that BridgeToken smart contracts have high-level protections in place to be safe from most attacks.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Fishing Town is A blockchain game inspired by fishing life and a passion for aquatic animals with old-school art pixel design. "Play-Fun-Earn", the main theme of the game, means that the game will maintain the balance of playing and earning which will be the part that every player is looking forward to.

There are three parts of the BridgeToken scope: BridgeToken, RodPresale, and RodShop.

BridgeToken is a mechanism that is used for transferring tokens between off-chain and on-chain.

RodPresale and RodShop are the places that allow users to buy Rod NFT. Users can buy the Rod NFT from RodPresale at a pre-sale price and buy the Rod NFT from the RodShop using any token defined by the owner.

Scope Information:

Project Name	BridgeToken
Website	https://fishingtown.io/
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Nov 30, 2021 - Dec 1, 2021
Reassessment Date	Dec 15, 2021

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 5e5a33244458407917f12ba9b1bb0fe3ec1ce0af)

Contract	Location (URL)
BridgeBase	https://github.com/Fishingtown/FisihingTownBridgeToken/blob/5e5a332444/BridgeBase.sol
FishingTownGilBridge	https://github.com/Fishingtown/FisihingTownBridgeToken/blob/5e5a332444/FishingTownGilBridge.sol
FishingTownRodPresale	https://github.com/Fishingtown/FisihingTownBridgeToken/blob/5e5a332444/FishingTownRodPresale.sol
FishingTownRodShop	https://github.com/Fishingtown/FisihingTownBridgeToken/blob/5e5a332444/FishingTownRodShop.sol

Reassessment: (Commit: a5586ce1bec0ea755422500446318ab03e2e344b)

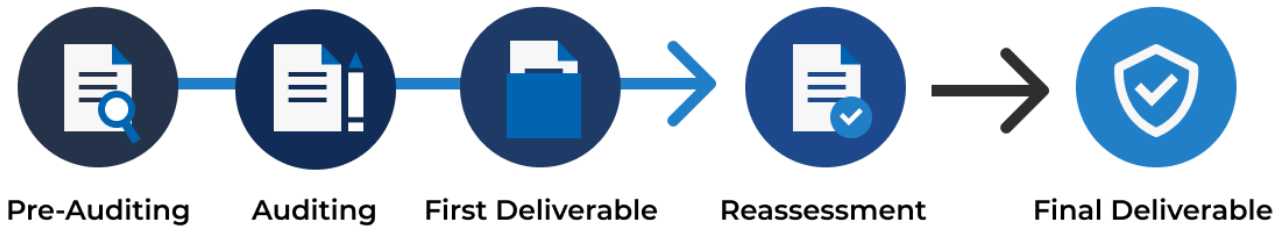
Contract	Location (URL)
BridgeBase	https://github.com/Fishingtown/FisihingTownBridgeToken/blob/a5586ce1be/BridgeBase.sol
FishingTownGilBridge	https://github.com/Fishingtown/FisihingTownBridgeToken/blob/a5586ce1be/FishingTownGilBridge.sol
FishingTownRodPresale	https://github.com/Fishingtown/FisihingTownBridgeToken/blob/a5586ce1be/FishingTownRodPresale.sol
FishingTownRodShop	https://github.com/Fishingtown/FisihingTownBridgeToken/blob/a5586ce1be/FishingTownRodShop.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Use of Upgradable Contract Design
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Improper Kill-Switch Mechanism

Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

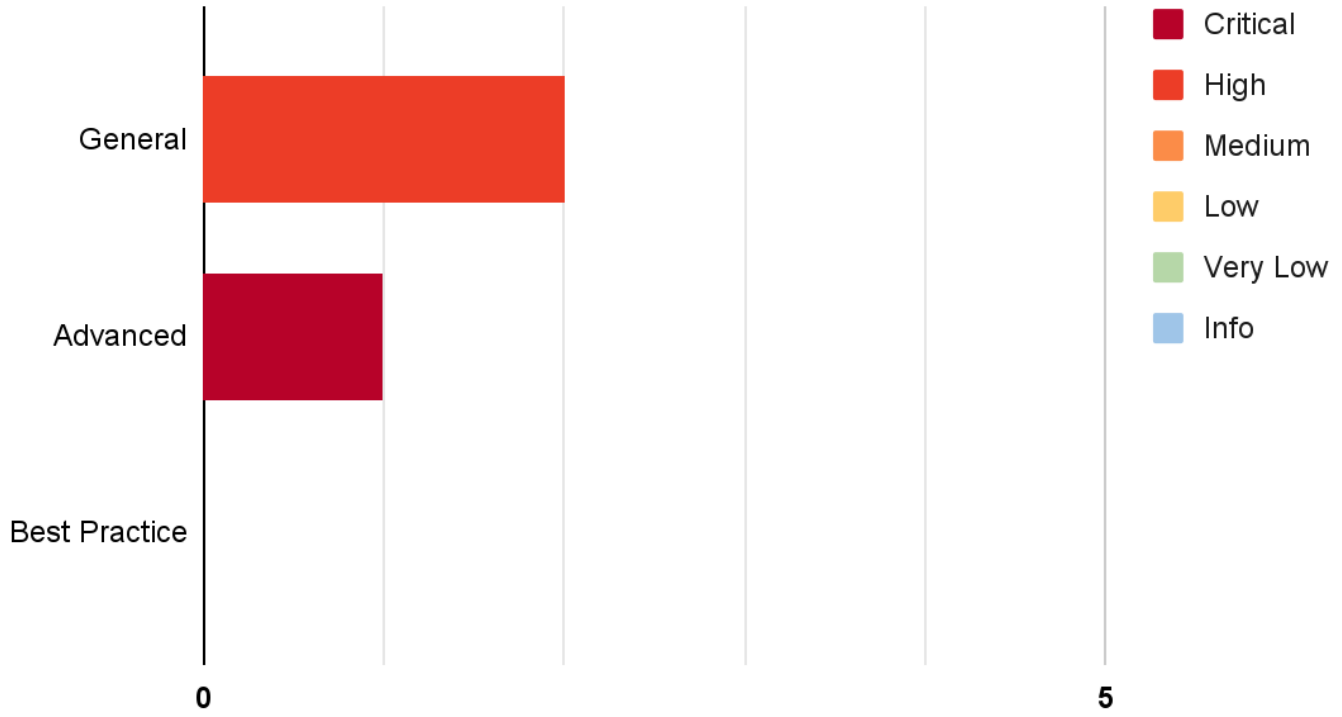
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	**Very Low**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

4. Summary of Findings

From the assessments, Inspex has found 3 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Use of Unsafe Price Source	Advanced	**Critical**	Resolved
IDX-002	Incorrect Token Ordering	General	**High**	Resolved
IDX-003	Improper Design for Operator Privilege	General	**High**	Resolved *

* The mitigations or clarifications by FishingTown can be found in Chapter 5.

5. Detailed Findings Information

5.1. Use of Unsafe Price Source

ID	IDX-001
Target	FishingTownRodShop
Category	Advanced Smart Contract Vulnerability
CWE	CWE-807: Reliance on Untrusted Inputs in a Security Decision
Risk	**Severity: Critical** **Impact: High** The $FHTN price on the platform can be arbitrarily manipulated by the flashloan attack to make the $FHTN overpriced. The users can use a tiny amount of the overpriced $FHTN to buy the fishing rod, which is unfair to other users. This issue can cause monetary and reputation damage to the platform. **Likelihood: High** It is very likely that the price will be manipulated because the attack is very profitable.
Status	Resolved FishingTown has resolved this issue by using TWAP oracle in commit `2dbff9e3449ace85d1f806baadadf77d6878163e`.

5.1.1. Description

The `fhtnUsdRate()` function return the real-time $FHTN price which is calculated by the reserves of $FHTN and another token from the PancakeSwap's pair contract, which can be manipulated easily by flashloan attack.

FishingTownRodShop.sol

```
72  function fhtnUsdRate() public view returns (uint256) {
73        (uint256 reserve0, uint256 reserve1, ) = pair.getReserves();
74        return ((reserve1 * 1 ether) / reserve0);
75  }
```

The attacker can use the flashswap feature of PancakeSwap to flashloan USD token from another pair and swap the flashloaned USD token to $FHTN to inflate the $FHTN price.

The return value of the `fhtnUsdRate()` function is used in the `currentPriceInFhtn()` as the price of $FHTN.

FishingTownRodShop.sol

```
63  function currentPriceInFhtn() public view returns (uint256) {
```

```
64      uint256 _fhtnUsdRate = fhtnUsdRate();
65      if (maxPriceInUsd < (priceInFhtn * _fhtnUsdRate) / (1 ether)) {
66          return (maxPriceInUsd * 1 ether) / _fhtnUsdRate;
67      }
68
69      return priceInFhtn;
70  }
```

The `currentPriceInFhtn()` is used to determine the amount of $FHTN required to purchase the Fishing Rod NFT.

FishingTownRodShop.sol

```
52  function purchaseFishingRod() external {
53      require(saleAmount < salesQuota, "sold out");
54      saleAmount++;
55
56      uint256 _currentPriceInFhtn = currentPriceInFhtn();
57      fhtn.safeTransferFrom(msg.sender, treasuryAddress, _currentPriceInFhtn);
58      uint256 tokenId = fishingRod.safeMint(msg.sender);
59
60      emit Purchase(msg.sender, tokenId, _currentPriceInFhtn);
61  }
```

If the inflated price of $FHTN is used to calculate the amount to purchase the NFT in the same transaction, the value of $FHTN will be massively inflated, allowing the attacker to use a little amount of $FHTN to purchase the NFT.

5.1.2. Remediation

Inspex suggests using the price data from a trustable price oracle provider.

If the price of the needed assets are not available from any trustable sources, Inspex suggests using time-weighted average price[2] instead of directly quoting from the reserves.

5.2. Incorrect Token Ordering

ID	IDX-002
Target	FishingTownRodShop
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity: High** **Impact: High** The users can purchase the NFT with the miscalculated $FHTN price. If the $FHTN price is inflated from the miscalculation, the amount of $FHTN to purchase the NFT will be lower, resulting in the platform receiving less $FHTN than it should be. **Likelihood: Medium** The addresses of `token0` and `token1` are ordered by the address value. The $FHTN contract is not deploy yet, so the address value of $FHTN can be higher or lower than another token.
Status	**Resolved** FishingTown has resolved this issue by removing the `fhtnUsdRate()` function and switching to TWAP oracle in commit `2dbff9e3449ace85d1f806baadadf77d6878163e`.

5.2.1. Description

The `fhtnUsdRate()` function is used to calculate the $FHTN price. The price depends on the reserve amounts of $FHTN and USD token in the PancakeSwap's pair contract.

FishingTownRodShop.sol

```
72  function fhtnUsdRate() public view returns (uint256) {
73      (uint256 reserve0, uint256 reserve1, ) = pair.getReserves();
74      return ((reserve1 * 1 ether) / reserve0);
75  }
```

The calculation can be incorrect when the `reserve1` is not $FHTN because the pair contract orders the token by address value. The lower address value is the `token0` and another is `token1` when the pair is created. [3]

```
25  function createPair(address tokenA, address tokenB) external returns (address
    pair) {
26      require(tokenA != tokenB, 'Pancake: IDENTICAL_ADDRESSES');
27      (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) :
    (tokenB, tokenA);
28      require(token0 != address(0), 'Pancake: ZERO_ADDRESS');
```

```
29      require(getPair[token0][token1] == address(0), 'Pancake: PAIR_EXISTS'); //
    single check is sufficient
30      bytes memory bytecode = type(PancakePair).creationCode;
31      bytes32 salt = keccak256(abi.encodePacked(token0, token1));
32      assembly {
33          pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
34      }
35      IPancakePair(pair).initialize(token0, token1);
36      getPair[token0][token1] = pair;
37      getPair[token1][token0] = pair; // populate mapping in the reverse
    direction
38      allPairs.push(pair);
39      emit PairCreated(token0, token1, pair, allPairs.length);
40  }
```

The `reserve1` parameter in the `fhtnUsdRate()` function refers to the reserve of `token0` in the pair contract.

At the time of the audit, the pair of $FHTN token contract is not deployed yet, and the address of the $FHTN token contract is unknown. So, it is possible that the order of `token0` and `token1` can be incorrect.

5.2.2. Remediation

Inspex suggests adding the condition to check whether `token0` or `token1` is $FHTN, for example:

FishingTownRodShop.sol

```
72  function fhtnUsdRate() public view returns (uint256) {
73      (uint256 reserve0, uint256 reserve1, ) = pair.getReserves();
74      return pair.token0 == address(fhtn) ? (reserve1 * 1 ether) / reserve0 :
    (reserve0 * 1 ether) / reserve1;
75  }
```

5.3. Improper Design for Operator Privilege

ID	IDX-003
Target	BridgeBase
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The permission to mint the `_token` is controlled by the owner address of the `BridgeBase` contract, which is the central point of minting from the off-chain server. The private key holder of the owner address can arbitrarily mint the `_token`, which can be unfair to the `_token` holders. This causes monetary damage to the `_token` holders and reputation damage to the platform. **Likelihood: Medium** Only the holder of the owner address private key can mint the `_token`, but there is no restriction to prevent this action from being done excessively.
Status	Resolved * The FishingTown team has mitigated this issue by implementing a consensus mechanism to verify the minting transactions. The minting transaction requires the signing of the majority of trusted validators. The validators will compare the off-chain burning amount with the on-chain minting amount to make sure that the amount is correct before signing the transaction. The token minting transaction requires at least half of the trusted validators, and no lesser than 3 validators, to validate the transaction. The `grantTrustedValidator()` and the `revokeTrustedValidator()` functions are added from this fixing. These functions are used to add and remove validators from the list and can be called anytime by the owner. However, the FishingTown team has confirmed that they will implement the timelock mechanism when deploying the smart contract to the mainnet. The users will be able to monitor the timelock for the changes on the smart contract and act accordingly if it is being misused. At the time of the reassessment, the contracts are not deployed yet, so the use of timelock is not confirmed. For the platform users, please verify that the timelock is properly deployed and the validators are trusted by the user before using this platform. The mitigation is applied in commit `a5586ce1bec0ea755422500446318ab03e2e344b`.

5.3.1. Description

In the `BridgeBase` contract, the owner can use the `transferTokenToOnchain()` function, to mint the token to any recipient. This privileged function has a very high impact as the total token supply can be controlled by the owner address that can mint unlimited amounts to any address.

BridgeBase.sol

```
30 function transferTokenToOnchain(
31     uint256 offchainNonce,
32     address to,
33     uint256 amount
34 ) external onlyOwner {
35     require(offchainNonces[offchainNonce] == false, "nonce already used");
36     offchainNonces[offchainNonce] = true;
37     token.mint(to, amount);
38
39     emit TransferTokenToOnchain(msg.sender, offchainNonce, to, amount);
40 }
```

From the current design, the owner is a wallet address with the private key controlled by FishingTown's off-chain server. Therefore, the holder of the owner's private key can freely mint the funds from the `BridgeBase` contract. If the mechanism to mint `_token` from the off-chain server is working incorrectly or can be controlled by a malicious actor, the minting in the `BridgeBase` contract will be incorrect too.

5.3.2. Remediation

Inspex suggests implementing a consensus mechanism to verify that the transaction is approved by the majority of trusted validators before minting or burning the token from the `BridgeBase` contract to prevent unverified actions from being done.

For example, this can be done by storing a list of trusted validators on the smart contract. The trusted validators then sign each transaction with a signature of their own, and submit them to the smart contract via the operator. Those signatures can then be verified on the smart contract using `ecrecover()` function, comparing them with the list of trusted validators.

6. Appendix

6.1. About Inspex

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] “OWASP Risk Rating Methodology.” [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

[2] “TWAP Oracle” [Online]. Available: https://docs.uniswap.org/protocol/V2/concepts/core-concepts/oracles. [Accessed: 01-November-2021]

[3] “Pancake Swap” [Online] .Available: https://github.com/pancakeswap/pancake-swap-core/blob/master/contracts/PancakeFactory.sol#L27. [Accessed: 22-December-2021]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE