SMART CONTRACT AUDIT REPORT

for

FishingTown Tokens

Prepared By: Yiqun Chen

PeckShield

December 13, 2021

Document Properties

Client	FishingTown
Title	Smart Contract Audit Report
Target	FishingTown Tokens
Version	1.1
Author	Patrick Liu
Auditors	Patrick Liu, Xuxian Jiang
Reviewed by	Yiqun Chen
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author	Description
1.1	December 13, 2021	Patrick Liu	Final Release (Amended #1)
1.0	December 4, 2021	Patrick Liu	Final Release

Contact

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

Name	Yiqun Chen
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related source code of the FishingTown, FishingTownGilToken and FishingTownRod contracts, we outline in the report our systematic method to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistency between smart contract code and the documentation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of the smart contracts shows no ERC20/ERC721-compliance issues, though there exist certain privileged roles that may govern the token operations. This document outlines our audit results.

1.1 About FishingTown Tokens

Fishing Town is a fishing game based on the BSC blockchain. It takes classic fishing games to the next level by featuring unique, one-of-a-kind NFT assets in the form of fishing rods and fishes. Players can collect these special fishing rods, catch, and sell rare fishes to earn rewards. The main tokens in the game are FHTN. In addition, it has the secondary tokens GIL which are used for specific purposes. Both FHTN and GIL are ERC20-compliant tokens. FHTR is the ERC721-compliant token which is used to represent NFT assets in the game.

The basic information of the audited smart contracts is as follows:

Table 1.1: Basic Information of FishingTown Tokens

Item	Description
Issuer	FishingTown
Website	https://fishingtown.io/
Type	Ethereum ERC20/ERC721 Token Contracts
Platform	Solidity
Audit Method	Whitebox
Audit Completion Date	December 13, 2021

In the following, we show the git repository and the commit hash value used in this audit:

- https://github.com/Fishingtown/FishingTownToken (2506330)

And this is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/Fishingtown/FishingTownToken (c2a5995)

1.2 About PeckShield

PeckShield Inc. [4] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystem by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [3]:

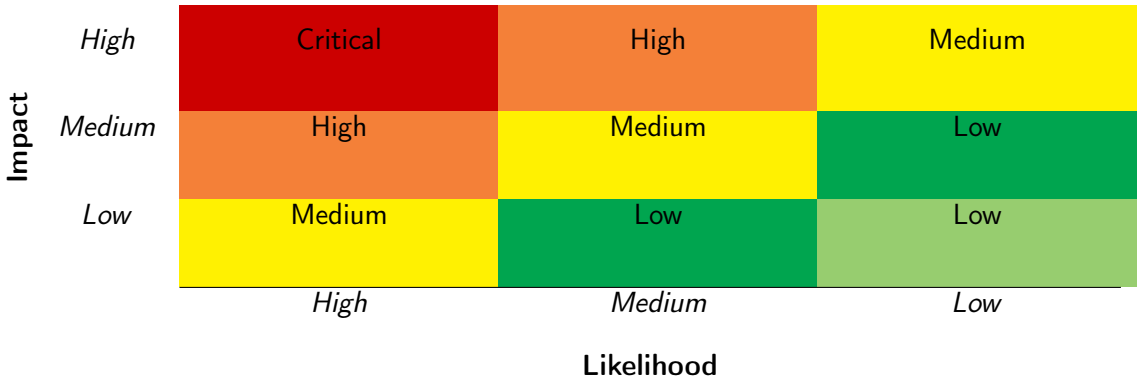
- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk;

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

We perform the audit according to the following procedures:

- 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.
- ERC20 Compliance Checks: We then manually check whether the implementation logic of the audited smart contract(s) follows the standard ERC20 specification and other best practices.
- ERC721 Compliance Checks: We then manually check whether the implementation logic of the audited smart contract(s) follows the standard ERC721 specification and other best practices.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Table 1.2: Vulnerability Severity Classification



To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
	Approve / TransferFrom Race Condition
ERC20 Compliance Checks	Compliance Checks (Section 3)
ERC721 Compliance Checks	Compliance Checks (Section 4)
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

2 Findings

2.1 Summary

Here is a summary of our findings after analyzing the FishingTown token contract. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place ERC20 and ERC721 related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	1	■
Low	0	
Informational	0	
Total	1	

Moreover, we explicitly evaluate whether the given contracts follow the standard ERC20 or ERC721 specification and other known best practices, and validate its compatibility with other similar tokens and current DeFi protocols. The detailed ERC20 compliance checks are reported in Section 3 for FHTN and GIL tokens. The detailed ERC721 compliance checks are reported in Section 4 for FHTR token. After that, we examine one issue that needs to be brought up and paid more attention to. (The finding is categorized in the above table.) Additional information can be found in the next subsection, and the detailed discussions are in Section 5.

2.2 Key Findings

Overall, no ERC20 or ERC721 compliance issue was found, and our detailed checklist can be found in Sections 3 and 4. However, the smart contract implementation can be improved because of the existence of 1 medium-severity vulnerability.

Table 2.1: Key FishingTown Tokens Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Medium	Trust Issue Of Admin Roles	Security Features	Mitigated

Please refer to Sections 3 and 4 for our detailed compliance checks and Section 5 for elaboration of the reported issue.

3 | ERC20 Compliance Checks

The ERC20 specification defines a list of API functions (and relevant events) that each token contract is expected to implement (and emit). The failure to meet these requirements means the token contract cannot be considered to be ERC20-compliant. Naturally, as the first step of our audit, we examine the list of API functions defined by the ERC20 specification and validate whether there exist any inconsistency or incompatibility in the implementation or the inherent business logic of the audited contract(s).

Table 3.1: Basic `View-Only` Functions Defined in The ERC20 Specification

Item	Description	Status
name()	Is declared as a public view function	✓
	Returns a string, for example "Tether USD"	✓
symbol()	Is declared as a public view function	✓
	Returns the symbol by which the token contract should be known, for example "USDT". It is usually 3 or 4 characters in length	✓
decimals()	Is declared as a public view function	✓
	Returns decimals, which refers to how divisible a token can be, from 0 (not at all divisible) to 18 (pretty much continuous) and even higher if required	✓
totalSupply()	Is declared as a public view function	✓
	Returns the number of total supplied tokens, including the total minted tokens (minus the total burned tokens) ever since the deployment	✓
balanceOf()	Is declared as a public view function	✓
	Anyone can query any address' balance, as all data on the blockchain is public	✓
allowance()	Is declared as a public view function	✓
	Returns the amount which the spender is still allowed to withdraw from the owner	✓

Our analysis shows that there is no ERC20 inconsistency or incompatibility issue found in the audited `FHTN` and `GIL` tokens. In the surrounding two tables, we outline the respective list of basic `view-only` functions (Table 3.1) and key `state-changing` functions (Table 3.2) according to the widely-

Table 3.2: Key `State-Changing` Functions Defined in The ERC20 Specification

Item	Description	Status
transfer()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the caller does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring to zero address	✓
transferFrom()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the spender does not have enough token allowances to spend	✓
	Updates the spender's token allowances when tokens are transferred successfully	✓
	Reverts if the from address does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring from zero address	✓
	Reverts while transferring to zero address	✓
approve()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token approval status	✓
	Emits Approval() event when tokens are approved successfully	✓
	Reverts while approving to zero address	✓
Transfer() event	Is emitted when tokens are transferred, including zero value transfers	✓
	Is emitted with the from address set to $address(0x0)$ when new tokens are generated	✓
Approval() event	Is emitted on any successful call to approve()	✓

adopted ERC20 specification. In addition, we perform a further examination on certain features that are permitted by the ERC20 specification or even further extended in follow-up refinements and enhancements (e.g., ERC777/ERC2222), but not required for implementation. These features are generally helpful, but may also impact or bring certain incompatibility with current DeFi protocols. Therefore, we consider it is important to highlight them as well. This list is shown in Table 3.3.

Table 3.3: Additional `Opt-in` Features Examined in Our Audit

Feature	Description	Opt-in
Deflationary	Part of the tokens are burned or transferred as fee while on transfer()/transferFrom() calls	—
Rebasing	The balanceOf() function returns a re-based balance instead of the actual stored amount of tokens owned by the specific address	—
Pausable	The token contract allows the owner or privileged users to pause the token transfers and other operations	—
Blacklistable	The token contract allows the owner or privileged users to blacklist a specific address such that token transfers and other operations related to that address are prohibited	—
Mintable	The token contract allows the owner or privileged users to mint tokens to a specific address	✓
Burnable	The token contract allows the owner or privileged users to burn tokens of a specific address	✓

4 | ERC721 Compliance Checks

The ERC721 standard for non-fungible tokens, also known as deeds. Inspired by the ERC-20 token standard, the ERC721 specification defines a list of API functions (and relevant events) that each token contract is expected to implement (and emit). The failure to meet these requirements means the token contract cannot be considered to be ERC721-compliant. Naturally, we examine the list of necessary API functions defined by the ERC721 specification and validate whether there exist any inconsistency or incompatibility in the implementation or the inherent business logic of the audited contract(s).

Table 4.1: Basic `View-Only` Functions Defined in The ERC721 Specification

Item	Description	Status
balanceOf()	Is declared as a public view function	✓
	Anyone can query any address' balance, as all data on the blockchain is public	✓
ownerOf()	Is declared as a public view function	✓
	Returns the address of the owner of the NFT	✓
getApproved()	Is declared as a public view function	✓
	Reverts while '_tokenId' does not exist	✓
	Returns the approved address for this NFT	✓
isApprovedForAll()	Is declared as a public view function	✓
	Returns a boolean value which check '_operator' is an approved operator	✓

Our analysis shows that there is no ERC721 inconsistency or incompatibility issue found in the audited `FHTR` token. In the surrounding two tables, we outline the respective list of basic `view-only` functions (Table 4.1) and key `state-changing` functions (Table 4.2) according to the widely-adopted ERC721 specification.

Table 4.2: Key `State-Changing` Functions Defined in The ERC721 Specification

Item	Description	Status
safeTransferFrom()	Is declared as a public function	✓
	Reverts while 'to' refers to a smart contract and not implement IERC721Receiver-onERC721Received	✓
	Reverts unless 'msg.sender' is the current owner, an authorized operator, or the approved address for this NFT	✓
	Reverts while '_tokenId' is not a valid NFT	✓
	Reverts while '_from' is not the current owner	✓
	Reverts while transferring to zero address	
	Emits Transfer() event when tokens are transferred successfully	✓
transferFrom()	Is declared as a public function	✓
	Reverts unless 'msg.sender' is the current owner, an authorized operator, or the approved address for this NFT	✓
	Reverts while '_tokenId' is not a valid NFT	✓
	Reverts while '_from' is not the current owner	✓
	Reverts while transferring to zero address	
	Emits Transfer() event when tokens are transferred successfully	✓
approve()	Is declared as a public function	✓
	Reverts unless 'msg.sender' is the current owner, an authorized operator, or the approved address for this NFT	✓
	Emits Approval() event when tokens are approved successfully	✓
setApprovalForAll()	Is declared as a public function	✓
	Reverts while not approving to caller	✓
	Emits ApprovalForAll() event when tokens are approved successfully	✓
Transfer() event	Is emitted when tokens are transferred	✓
Approval() event	Is emitted on any successful call to approve()	✓
ApprovalForAll() event	Is emitted on any successful call to setApprovalForAll()	✓

5 Detailed Results

5.1 Trust Issue Of Admin Roles

- ID: PVE-001
- Severity: Medium
- Likelihood: Low
- Impact: High
- Target: `Multiple Contracts`
- Category: Security Features [2]
- CWE subcategory: CWE-287 [1]

Description

In the `FishingTown` and `FishingTownGilToken` token contracts, there is a privileged `admin` account (with the role of `DEFAULT_ADMIN_ROLE` that is assigned in the `constructor()`) in each token contract that plays a critical role in assigning roles to other accounts which may be in the position of governing and regulating the token-related operations (e.g., mint new tokens to specified account).

To elaborate, we show below the `mint()` function in the `FishingTown` contract. The `mint()` function allows the `MINTER` to add up to $100,000,000$ FHTN tokens into circulation and the recipient can be directly provided when the mint operation takes place (line 18).

```
9  contract FishingTown is ERC20, ERC20Burnable, AccessControl, ERC20Permit {
10     bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
11     uint256 private _cap = 100000000 ether;
12
13     constructor() ERC20("FishingTown", "FHTN") ERC20Permit("FishingTown") {
14         _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
15         _setupRole(MINTER_ROLE, msg.sender);
16     }
17
18     function mint(address to, uint256 amount) external onlyRole(MINTER_ROLE) {
19         _mint(to, amount);
20     }
21
22     function cap() public view returns (uint256) {
23         return _cap;
24     }
```

```
25
26      function _mint(address account, uint256 amount) internal override {
27          require(ERC20.totalSupply() + amount <= cap(), "ERC20: cap exceeded");
28          super._mint(account, amount);
29      }
30  }
```

Listing 5.1: `FishingTown::mint()`

The extra power to the minter is a counter-party risk to current contract users. It is worrisome if the granted `MINTER` account is a plain EOA account. Note that a multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed DAO.

The same issue is also applicable to the `FishingTownGilToken` contract. The difference is that the `GIL` token contract allows the `MINTER` to add unrestricted tokens into circulation.

Recommendation Promptly transfer the privileged account to the intended `DAO`-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status This issue has been mitigated by introducing the `timelock` mechanism in the following commit: `c2a5995`.

6 | Conclusion

In this security audit, we have examined the design and implementation of the `FishingTown Token` contracts. During our audit, we first checked all respects related to the compatibility of the ERC20 and ERC721 specification and other known ERC Standard pitfalls/vulnerabilities. We then proceeded to examine other areas such as coding practices and business logics. Overall, although no critical or high level vulnerabilities were discovered, we identified one issue. In the meantime, as disclaimed in Section 1.4, we appreciate any constructive feedbacks or suggestions about our findings, procedures, audit scope, etc.

References

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

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

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

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