SMART CONTRACT AUDIT REPORT

for

AI Waifu

Prepared By: Xiaomi Huang

PeckShield

March 5, 2024

Document Properties

Client	AI Waifu
Title	Smart Contract Audit Report
Target	AI Waifu
Version	1.0
Author	Xuxian Jiang
Auditors	Jason Shen, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	March 5, 2024	Xuxian Jiang	Final Release
1.0-rc	February 21, 2024	Xuxian Jiang	Release Candidate #1

Contact

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

Name	Xiaomi Huang
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 AI Waifu protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About AI Waifu

AI Waifu is designed to be a Waifu companion game. The players get to discover their Waifu backstory, unlock NSFW content, flirt with other Waifus (PvP), and protect your Waifu through strategic resources, as well as claim rewards for $WAIFU tokens. The basic information of audited contracts is as follows:

Table 1.1: Basic Information of AI Waifu

Item	Description
Name	AI Waifu
Website	https://aiwaifu.gg/
Type	Smart Contract
Language	Solidity
Audit Method	Whitebox
Latest Audit Report	March 5, 2024

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit:

- https://github.com/aiwaifu-gg/waifu-contracts.git (0074ced)

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

- https://github.com/aiwaifu-gg/waifu-contracts.git (1b21c18)

1.2 About PeckShield

PeckShield Inc. [10] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems 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).

Table 1.2: Vulnerability Severity Classification

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

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [9]:

- 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 accordingly classified into four categories, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

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 or analysis 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

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
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
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

additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- 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.

- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.

- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [8], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings. Moreover, in case there is an issue that may affect an active protocol that has been deployed, the public version of this report may omit such issue, but will be amended with full details right after the affected protocol is upgraded with respective fixes.

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 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the design and implementation of the `AI Waifu` protocol smart contracts. 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 DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	2	
Low	3	
Informational	0	
Total	5	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 2 medium-severity vulnerabilities and 3 low-severity vulnerabilities.

Table 2.1: Key Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Improved Validation on Protocol Parameters in WaifuToken	Coding Practices	Resolved
PVE-002	Low	Revisited Ownable Inheritance in Shop	Business Logic	Resolved
PVE-003	Low	Redundant _taxProcessing() Handling in WaifuToken	Coding Practices	Resolved
PVE-004	Medium	Confused Defender Account in GameManager::tempt()	Business Logic	Resolved
PVE-005	Medium	Trust Issue Of Admin Keys	Security Features	Mitigated

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Improved Validation on Protocol Parameters in WaifuToken

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `WaifuToken`, `Shop`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1126 [1]

Description

DeFi protocols typically have a number of system-wide parameters that can be dynamically configured on demand. The `AI Waifu` protocol is no exception. Specifically, if we examine the `WaifuToken` contract, it has defined a number of protocol-wide risk parameters, such as `projectBuyTaxBasisPoints` and `projectSellTaxBasisPoints`. In the following, we show the corresponding constructor routine that initializes their values.

```
    function setProjectTaxRates(
        uint16 newProjectBuyTaxBasisPoints_,
        uint16 newProjectSellTaxBasisPoints_
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        uint16 oldBuyTaxBasisPoints = projectBuyTaxBasisPoints;
        uint16 oldSellTaxBasisPoints = projectSellTaxBasisPoints;

        projectBuyTaxBasisPoints = newProjectBuyTaxBasisPoints_;
        projectSellTaxBasisPoints = newProjectSellTaxBasisPoints_;

        emit ProjectTaxBasisPointsChanged(
            oldBuyTaxBasisPoints,
            newProjectBuyTaxBasisPoints_,
            oldSellTaxBasisPoints,
            newProjectSellTaxBasisPoints_
        );
    }
```

Listing 3.1: WaifuToken::setProjectTaxRates()

These parameters define various aspects of the protocol operation and maintenance and need to exercise extra care when configuring or updating them. Our analysis shows the update logic on these parameters can be improved by applying more rigorous sanity checks. Based on the current implementation, certain corner cases may lead to an undesirable consequence. For example, we can improve the above setter by further enforcing the following requirements: `require(newProjectBuyTaxBasisPoints_ < BP_DENOM)` and `require(newProjectSellTaxBasisPoints_ < BP_DENOM)`. In addition, there is a need to set `_tokenHasTax = false` when `newProjectBuyTaxBasisPoints_==0 && newProjectSellTaxBasisPoints_==0`.

Recommendation Validate any changes regarding these system-wide parameters to ensure they fall in an appropriate range.

Status This issue has been fixed in the following commit: `1b21c18`.

3.2 Revisited Ownable Inheritance in Shop

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `Shop`
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

In `AI Waifu`, there is a core `Shop` contract that implements the `Uniswap`-like `DEX` engine with the support of `ERC1155` standard tokens. While examining the `Shop` contract, we notice its inheritance from `Ownable` that provides a basic access control mechanism, where a privileged account (i.e., owner) can be granted exclusive access to specific functions. However, our analysis shows that there is no function in `Shop` that has been defined to make use of this access control mechanism.

```
contract Shop is ReentrancyGuard, IShop, Ownable, ERC1155, ERC1155Burnable {
    // Variables
    IERC1155 internal immutable token; // address of the ERC-1155 token contract
    address internal immutable currency; // address of the ERC-20 currency used for exchange
    address internal immutable factory; // address for the factory that created this contract

    // Royalty variables
    bool internal immutable IS_ERC2981; // whether token contract supports ERC-2981
    ...
}
```

Listing 3.2: The `Shop` Contract

To elaborate, we show above the code snippet of this Shop contract. The Ownable inheritance is unnecessary and can be safely removed. From another perspective, there is a possibility of making use of the access control mechanism to uncover funds that may be accidentally sent to the contract.

Recommendation Revise the above contract to remove the Ownable inheritance.

Status This issue has been fixed in the following commit: 1b21c18.

3.3 Redundant _taxProcessing() Handling in WaifuToken

- ID: PVE-003
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: WaifuToken
- Category: Coding Practices [6]
- CWE subcategory: CWE-563 [3]

Description

The AI Waifu protocol makes good use of a number of reference contracts, such as ERC20Permit, AccessControl, EnumerableSet, and SafeERC20, to facilitate its code implementation and organization. For example, the WaifuToken smart contract has so far imported at least five reference contracts. However, we observe the inclusion of certain unused code or the presence of unnecessary redundancies that can be safely removed.

For example, if we examine closely the _taxProcessing() routine, it charges the buy/sell tax for the buy/sell transactions. In particular, when it is a sell operation, the conditions of sLiquidityPool(to_)&& totalSellTaxBasisPoints()> 0) (line 188) are met, which makes the following if (projectSellTaxBasisPoints > 0) condition (line 189) unnecessary. A similar redundancy is also observed for the buy operation (line 200).

```
176     function _taxProcessing(
177         bool applyTax_,
178         address to_,
179         address from_,
180         uint256 sentAmount_
181     ) internal returns (uint256 amountLessTax_) {
182         amountLessTax_ = sentAmount_;
183         unchecked {
184             if (_tokenHasTax && applyTax_) {
185                 uint256 tax;
186 
187                 // on sell
188                 if (isLiquidityPool(to_) && totalSellTaxBasisPoints() > 0) {
189                     if (projectSellTaxBasisPoints > 0) {
190                         uint256 projectTax = ((sentAmount_ *
```

```
                            projectSellTaxBasisPoints) / BP_DENOM);
                        projectTaxPendingSwap += uint128(projectTax);
                        tax += projectTax;
                    }
                }
                // on buy
                else if (
                    isLiquidityPool(from_) && totalBuyTaxBasisPoints() > 0
                ) {
                    if (projectBuyTaxBasisPoints > 0) {
                        uint256 projectTax = ((sentAmount_ *
                            projectBuyTaxBasisPoints) / BP_DENOM);
                        projectTaxPendingSwap += uint128(projectTax);
                        tax += projectTax;
                    }
                }

                if (tax > 0) {
                    _increaseBalance(address(this), tax);
                    emit Transfer(from_, address(this), tax);
                    amountLessTax_ -= tax;
                }
            }
        }
        return (amountLessTax_);
    }
```

Listing 3.3: `WaifuToken::_taxProcessing()`

Recommendation Consider the removal of the redundant state (or code) with a simplified, consistent implementation.

Status This issue has been fixed in the following commit: `1b21c18`.

3.4 Confused Defender Account in GameManager::tempt()

- ID: PVE-004
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `GameManager`
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

In `AI Waifu`, there is another core `GameManager` contract that implements various interactions with `Waifus`. There is a special type of interaction called `tempt` and our analysis shows its logic should be revisited.

To elaborate, we show below the related `tempt()` routine. Notice that there are two roles in it: `Temptor` and `Resistor Waifu`. `Temptor` is the one who tempts other `Waifus` and `Waifus` that are being tempted are known as `Resistor Waifu`. Our analysis shows the `defendAmount` state should be computed based on the `waifuId` owner (`IERC721(waifuNft).ownerOf(waifuId)`), not the `account` (`msg.sender`). Otherwise, as long as the owner removes the `isApprovedForAll` flag, no temptation will be successful.

```
    function tempt(uint256 waifuId, uint256 wager) external isActive(waifuId) {
        address account = _msgSender();
        require(
            _cooldownByWaifu[waifuId] < block.timestamp,
            "Waifu is on cooldown"
        );
        require(
            _cooldownByAddress[account] < block.timestamp,
            "Account is on cooldown"
        );
        IAIWaifu.Waifu memory waifu = IAIWaifu(waifuNft).waifu(waifuId);
        ERC1155Burnable ingredientContract = ERC1155Burnable(ingredientNft);
        uint256 temptIngredientId = temptMap[waifu.ingredientId];
        require(
            ingredientContract.balanceOf(account, temptIngredientId) >= wager,
            "Insufficient ingredient"
        );
        uint256 temptId = _nextTemptId++;
        ingredientContract.burn(account, temptIngredientId, wager);

        uint256 defendAmount = ingredientContract.isApprovedForAll(
            account,
            address(this)
        )
            ? ingredientContract.balanceOf(account, waifu.ingredientId)
            : 0;
        if (defendAmount > 0) {
            ingredientContract.burn(
                IERC721(waifuNft).ownerOf(waifuId),
                waifu.ingredientId,
                Math.min(defendAmount, wager)
            );
        }

        _cooldownByWaifu[waifuId] = block.timestamp + defendCooldown;
        _cooldownByAddress[account] = block.timestamp + temptCooldown;

        emit Tempted(account, waifuId, temptId, wager, defendAmount);
    }
```

Listing 3.4: `GameManager::tempt()`

Recommendation Revise the above logic to properly implement the temptation logic.

Status This issue has been fixed in the following commit: `b549e77`.

3.5 Trust Issue of Admin Keys

- ID: PVE-005
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `Multiple Contracts`
- Category: Security Features [5]
- CWE subcategory: CWE-287 [2]

Description

In the `AI Waifu` protocol, there is a privileged account (with the `DEFAULT_ADMIN_ROLE` role) that plays a critical role in governing and regulating the system-wide operations (e.g., configuring various parameters and managing pools). It also has the privilege to control or govern the flow of assets managed by this protocol. Our analysis shows that the privileged account needs to be scrutinized. In the following, we examine the privileged account and the related privileged accesses in current contracts.

```
    function addLiquidityPool(
        address newLiquidityPool_
    ) public onlyRole(DEFAULT_ADMIN_ROLE) {
        // Don't allow calls that didn't pass an address:
        if (newLiquidityPool_ == address(0)) {
            _revert(LiquidityPoolCannotBeAddressZero.selector);
        }
        // Only allow smart contract addresses to be added, as only these can be pools:
        if (newLiquidityPool_.code.length == 0) {
            _revert(LiquidityPoolMustBeAContractAddress.selector);
        }
        // Add this to the enumerated list:
        _liquidityPools.add(newLiquidityPool_);
        emit LiquidityPoolAdded(newLiquidityPool_);
    }

    function removeLiquidityPool(
        address removedLiquidityPool_
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        // Remove this from the enumerated list:
        _liquidityPools.remove(removedLiquidityPool_);
        emit LiquidityPoolRemoved(removedLiquidityPool_);
    }
    ...
    function withdrawERC20(
        address token_,
        uint256 amount_
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        if (token_ == address(this)) {
```

```
            _revert(CannotWithdrawThisToken.selector);
        }
        IERC20(token_).safeTransfer(_msgSender(), amount_);
    }
    ...
    function setProjectTaxRecipient(
        address projectTaxRecipient_
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        projectTaxRecipient = projectTaxRecipient_;
        emit ProjectTaxRecipientUpdated(projectTaxRecipient_);
    }

    function setProjectTaxRates(
        uint16 newProjectBuyTaxBasisPoints_,
        uint16 newProjectSellTaxBasisPoints_
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        uint16 oldBuyTaxBasisPoints = projectBuyTaxBasisPoints;
        uint16 oldSellTaxBasisPoints = projectSellTaxBasisPoints;

        projectBuyTaxBasisPoints = newProjectBuyTaxBasisPoints_;
        projectSellTaxBasisPoints = newProjectSellTaxBasisPoints_;

        emit ProjectTaxBasisPointsChanged(
            oldBuyTaxBasisPoints,
            newProjectBuyTaxBasisPoints_,
            oldSellTaxBasisPoints,
            newProjectSellTaxBasisPoints_
        );
    }
```

Listing 3.5: Example Privileged Functions in `WaifuToken`

Note that if the privileged `owner` account is a plain EOA account, this may be worrisome and pose counter-party risk to the exchange users. 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. In the meantime, a timelock-based mechanism can also be considered as mitigation.

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 as the team makes use of a multisig to act as the privileged owner.

4 Conclusion

In this audit, we have analyzed the design and implementation of the AI Waifu protocol, which is designed to be a Waifu companion game. The players get to discover their Waifu backstory, unlock NSFW content, flirt with other Waifus (PvP), and protect your Waifu through strategic resources, as well as claim rewards for $WAIFU tokens. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that Solidity-based smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-1126: Declaration of Variable with Unnecessarily Wide Scope. https://cwe.mitre.org/data/definitions/1126.html.

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

[3] MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

[4] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

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

[6] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[7] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[8] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

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

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