



# SMART CONTRACT AUDIT REPORT

for

## Xterio OnchainIAP



Prepared By: Xiaomi Huang

PeckShield  
July 22, 2024

## Document Properties

Client	Xterio
Title	Smart Contract Audit Report
Target	Xterio OnchainIAP
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	July 22, 2024	Xuxian Jiang	Final Release
1.0-rc1	July 20, 2024	Xuxian Jiang	Release Candidate #1

## Contact

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

Name	Xiaomi Huang
Phone	+86 156 0639 2692
Email	contact@peckshield.com

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	About Xterio . . . . .	4
1.2	About PeckShield . . . . .	5
1.3	Methodology . . . . .	5
1.4	Disclaimer . . . . .	7
<b>2</b>	<b>Findings</b>	<b>9</b>
2.1	Summary . . . . .	9
2.2	Key Findings . . . . .	10
<b>3</b>	<b>Detailed Results</b>	<b>11</b>
3.1	Inconsistent productExists Modifier Uses in OnchainIAP . . . . .	11
3.2	Improved Ether Transfer And Refund in OnchainIAP . . . . .	12
3.3	Trust Issue of Admin Keys . . . . .	13
<b>4</b>	<b>Conclusion</b>	<b>15</b>
	<b>References</b>	<b>16</b>

# 1 | Introduction

Given the opportunity to review the design document and related smart contract source code of new features in Xterio, 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 Xterio

Xterio is a Web3 gaming ecosystem & infrastructure, distinguishing itself as a gaming publisher with top-notch development skills and unparalleled distribution expertise. The audited OnchainIAP is the major functioning smart contract while Aggregator is an implementation of Chainlink's AggregatorV3Interface used for synchronizing oracle data from mainnet to other chains who are not supported by Chainlink. The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of Xterio OnchainIAP

Item	Description
Name	Xterio OnchainIAP
Type	Ethereum Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	July 22, 2024

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

- <https://github.com/XterioTech/xt-contracts.git> (ac374ff)

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

- <https://github.com/XterioTech/xt-contracts.git> (bc2ed2c)

## 1.2 About PeckShield

PeckShield Inc. [7] 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](mailto:contact@peckshield.com)).

Table 1.2: Vulnerability Severity Classification

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
		Likelihood		

## 1.3 Methodology

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [6]:

- 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.

To evaluate the risk, we go through a checklist of 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

Table 1.3: The Full Audit Checklist

Category	Checklist Items
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

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.

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) [5], 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 Logic	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 implementation of new features in Xterio. 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 logic, 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	1	
Low	2	
Informational	0	
Total	3	

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 1 medium-severity vulnerability and 2 low-severity vulnerabilities.

Table 2.1: Key Xterio OnchainIAP Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Inconsistent productExists Modifier Uses in OnchainIAP	Coding Practices	Resolved
PVE-002	Low	Improved Ether Transfer And Refund in OnchainIAP	Coding Practices	Resolved
PVE-003	Medium	Trust Issue of Admin Keys	Security Features	Mitigated

Besides 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 Inconsistent productExists Modifier Uses in OnchainIAP

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: OnchainIAP
- Category: Coding Practices [4]
- CWE subcategory: CWE-1126 [1]

#### Description

The Xterio protocol has a new feature OnchainIAP, which allows to manage the products and associated SKUs. While examining the management logic of products and associated SKUs, we notice the presence of a `productExists` modifier, which can be more consistently used.

In the following, we show the implementation of the related routines, i.e., `updateProductPaymentRecipient()` and `registerSKU()`. As their names indicate, they are used to configure the product's payment recipient as well as the product's SKUs. We notice the payment recipient and SKUs are associated with the product, which needs to be present for their configuration. Therefore, both functions should consistently have the `productExists(_productId)` modifier (line 115).

```

103     function updateProductPaymentRecipient(
104         uint32 _productId ,
105         address _paymentRecipient
106     ) external onlyRole(DEFAULT_ADMIN_ROLE) productExists(_productId) {
107         products[_productId].paymentRecipient = _paymentRecipient;
108     }
109
110     function registerSKU(
111         uint32 _productId ,
112         uint32 _skuld ,
113         uint256 _price ,
114         uint256 _amount
115     ) external onlyRole(DEFAULT_ADMIN_ROLE) {
116         products[_productId].skus[_skuld] = SKU(false , _amount, _price);
117

```

```

118     productSKUs[_productId].add(_skuld);
119 }

```

Listing 3.1: OnchainIAP::updateProductPaymentRecipient()/registerSKU()

**Recommendation** Add the `productExists(_productId)` modifier to related functions, including `registerSKU()`, `setDisableSKU()`, and `setPaymentMethodValid()`.

**Status** This issue has been fixed in the following commit: `bc2ed2c`.

## 3.2 Improved Ether Transfer And Refund in OnchainIAP

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: OnchainIAP
- Category: Coding Practices [4]
- CWE subcategory: CWE-1109 [1]

### Description

The OnchainIAP contract allows each product to have the flexibility in customizing the payment tokens for support. While reviewing the payment token support with the native coins, we notice its support may be improved.

To elaborate, we show below the code snippet of the `purchaseSKU()` routine, which allows to purchase a certain SKU with the native coin (e.g., `Ether`). We notice that this routine directly calls the native `send()` routine (lines 194 and 199) to transfer `Ether`. However, the `send()` is not recommended to use any more since the EIP-1884 may increase the gas cost and the 2300 gas limit may be exceeded. There is a helpful blog [stop-using-soliditys-transfer-now](#) that explains why the `send()` is not recommended any more.

```

189     if (_paymentTokenAddress == address(0)) {
190         require(
191             msg.value >= totalPrice,
192             "OnchainIAP: Insufficient payment"
193         );
194         bool success = payable(recipient).send(totalPrice);
195         require(success, "OnchainIAP: Transfer to recipient failed");
196
197         uint256 excess = msg.value - totalPrice;
198         if (excess > 0) {
199             bool refundSuccess = payable(msg.sender).send(excess);
200             require(refundSuccess, "OnchainIAP: Refund to sender failed");
201         }

```

202

}

Listing 3.2: OnchainIAP::purchaseSKU()

**Recommendation** Revisit the `purchaseSKU()` routine to transfer ETH using `call()`.

**Status** This issue has been fixed in the following commit: `bc2ed2c`.

### 3.3 Trust Issue of Admin Keys

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: Aggregator, OnchainIAP
- Category: Security Features [3]
- CWE subcategory: CWE-287 [2]

#### Description

In the Xterio protocol, there is a privileged account `owner` that plays a critical role in governing and regulating the system-wide operations (e.g., configure parameters, manager products/SKUs, and update oracles). 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.

```

93     function updateRoundData(
94         uint80 roundId,
95         int256 answer,
96         uint256 startedAt,
97         uint256 updatedAt
98     ) external onlyOwner {
99         require(
100             roundId > _latestRoundId,
101             "Round ID must be greater than latest"
102         );
103         _roundData[roundId] = RoundData(
104             roundId,
105             answer,
106             startedAt,
107             updatedAt,
108             roundId
109         );
110         _latestRoundId = roundId;
111     }
112
113     function updateAggregatorData(
114         uint8 decimals_,
115         string memory description_,
116         uint256 version_

```

```
117     ) external onlyOwner {  
118         _decimals = decimals_;  
119         _description = description_;  
120         _version = version_;  
121     }
```

Listing 3.3: Example Privileged Operations in `Aggregator`

We emphasize that the privilege assignment is necessary and consistent with the protocol design. However, it would be worrisome if the privileged 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. 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 plans the use of a multi-sig contract for the privileged account.



## 4 | Conclusion

In this audit, we have analyzed the design and implementation of new features in `xterio`, which is a web3 gaming ecosystem & infrastructure, distinguishing itself as a gaming publisher with top-notch development skills and unparalleled distribution expertise. The audited `OnchainIAP` is the major functioning smart contract while `Aggregator` is an implementation of Chainlink's `AggregatorV3Interface` used for synchronizing oracle data from mainnet to other chains who are not supported by Chainlink. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Moreover, 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 CATEGORY: 7PK - Security Features. <https://cwe.mitre.org/data/definitions/254.html>.
- [4] MITRE. CWE CATEGORY: Bad Coding Practices. <https://cwe.mitre.org/data/definitions/1006.html>.
- [5] MITRE. CWE VIEW: Development Concepts. <https://cwe.mitre.org/data/definitions/699.html>.
- [6] OWASP. Risk Rating Methodology. [https://www.owasp.org/index.php/OWASP\\_Risk\\_Rating\\_Methodology](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology).
- [7] PeckShield. PeckShield Inc. <https://www.peckshield.com>.