SMART CONTRACT AUDIT REPORT

for

AllDeFi

Prepared By: Xiaomi Huang

PeckShield

January 2, 2026

Document Properties

Client	AllDeFi
Title	Smart Contract Audit Report
Target	AllDeFi
Version	1.0
Author	Xuxian Jiang
Auditors	Matthew Jiang, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	January 2, 2026	Xuxian Jiang	Final Release
1.0-rc	January 2, 2026	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 AllDeFi 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 AllDeFi

AllDeFi is a compliant, institutional-grade yield protocol offering professional strategy management and institutional custody within a transparent, KYC-verified framework. The audited Vault facilitates user deposits through an epoch-based structure, with deposits processed in a controlled and secure manner.

Table 1.1: Basic Information of AllDeFi

Item	Description
Name	AllDeFi
Type	Smart Contract
Language	Solidity
Audit Method	Whitebox
Latest Audit Report	January 2, 2026

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

- https://github.com/alldefi/vault.git (996c9b4)

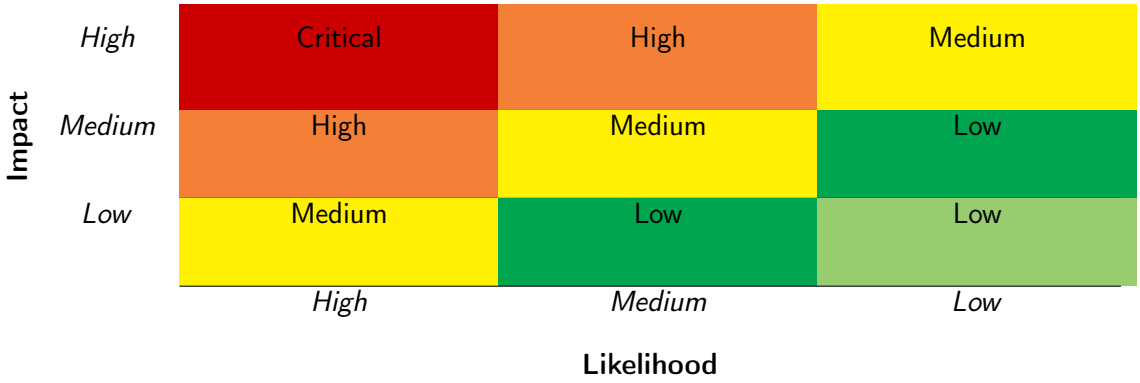
And here is the commit hash value after all fixes for the issues found in the audit have been applied:

- https://github.com/alldefi/vault.git (720aebf)

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).

Table 1.2: Vulnerability Severity Classification



1.3 Methodology

To standardize the evaluation, we define the following terminology based on 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 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

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

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 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 implementation of the `AllDeFi` protocol. 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	0	
Low	1	■
Informational	2	■■
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. 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, this smart contract is well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 low-severity vulnerability, and 2 informational suggestions.

Table 2.1: Key AllDeFi Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Informational	Suggested Emit of Parameter-Updating Events in initialize()	Coding Practice	Resolved
PVE-002	Informational	Improved epochId Update Logic in deposit()	Coding Practice	Resolved
PVE-003	Low	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 contract is being deployed on mainnet. Please refer to Section 3 for details.

3 Detailed Results

3.1 Suggested Emit of Parameter-Updating Events in initialize()

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Vault`
- Category: Coding Practices [4]
- CWE subcategory: CWE-1126 [1]

Description

In `Ethereum`, the `event` is an indispensable part of a contract and is mainly used to record a variety of runtime dynamics. In particular, when an `event` is emitted, it stores the arguments passed in transaction logs and these logs are made accessible to external analytics and reporting tools. `Events` can be emitted in a number of scenarios. One particular case is when system-wide parameters or settings are being changed. Another case is when tokens are being minted, transferred, or burned.

In the following, we use the `Vault`'s `initialize()` function as an example. It is designed to initialize a number of vault-wide risk parameters. While examining the events that reflect their changes, we notice the related events are not emitted yet. With that, we suggest to add `emit MinDepositUpdated(_minDeposit);`, `emit CustodianUpdated(_custodian);`, and `emit StageWindowsUpdated (_depositWindowDays, _processingWindowDays);` to reflect their changes.

```
64      function initialize(
65          string memory _name,
66          address _asset,
67          uint256 _minDeposit,
68          address _custodian,
69          uint64 _depositWindowDays,
70          uint64 _processingWindowDays
71      ) external initializer {
72          if (_asset == address(0)) revert ZeroAddress();
73          if (_custodian == address(0)) revert ZeroAddress();
74          if (_depositWindowDays == 0) revert InvalidDepositWindow();
```

```
76          __AccessControl_init();
77          __ReentrancyGuard_init();
78          __UUPSUpgradeable_init();

80          _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
81          _grantRole(KEEPER_ROLE, msg.sender);

83          name = _name;
84          asset = _asset;

86          minDeposit = _minDeposit;
87          custodian = _custodian;
88          depositWindowDays = _depositWindowDays;
89          processingWindowDays = _processingWindowDays;

91          epochStartedAt = uint64(block.timestamp);
92      }
```

Listing 3.1: `Vault::initialize()`

Recommendation Properly emit the respective event when a vault-side risk parameter is updated.

Status This issue has been fixed in the following commit: `720aebf`.

3.2 Improved epochId Update Logic in deposit()

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Vault`
- Category: Coding Practice [4]
- CWE subcategory: CWE-1126 [1]

Description

As mentioned earlier, the audited `Vault` contract facilitates user deposits through an epoch-based structure, with deposits processed in a controlled and secure manner. In the process of examining current deposit logic, we notice a minor improvement that can be made to only update the user deposit epoch when necessary.

In the following, we show the implementation of the related `deposit()` function, which handles the user deposit request. We notice this function always updates `userDeposit.epochId = currentEpochId ;` (line 214), which can be performed only when `userDeposit.epochId != currentEpochId`. In other words, it can be moved into the `else` branch (lines 210-212).

```
197     function deposit(uint256 amount) external payable nonReentrant {
198         if (!isWhitelisted[msg.sender]) revert NotWhitelisted();
199         uint64 currentEpochId = epochId;
200
201         Stage stage = vaultStage();
202         if (stage != Stage.Open) revert InvalidStage();
203         if (amount < minDeposit) revert LessThanMin();
204
205         Deposit storage userDeposit = _userDeposits[msg.sender];
206
207         // Update deposit info
208         if (userDeposit.epochId == currentEpochId) {
209             userDeposit.amount += amount;
210         } else {
211             // New epoch: refresh state to track current pending amount only.
212             userDeposit.amount = amount;
213         }
214         userDeposit.epochId = currentEpochId;
215
216         epochDeposits[currentEpochId] += amount;
217
218         if (asset == NATIVE_TOKEN) {
219             if (msg.value != amount) revert InvalidAmount();
220             _transferETH(custodian, amount);
221         } else {
222             if (msg.value > 0) revert InvalidAmount();
223             IERC20(asset).safeTransferFrom(msg.sender, custodian, amount);
224         }
225
226         emit Deposited(msg.sender, currentEpochId, amount);
227     }
```

Listing 3.2: `Vault::deposit()`

Recommendation Revise the above-mentioned routine to update user deposit epoch only when necessary.

Status This issue has been fixed in the following commit: `720aebf`.

3.3 Trust Issue of Admin Keys

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

Description

The `AllDeFi` protocol is designed with a privileged account (with the `DEFAULT_ADMIN_ROLE` role) that play a critical role in governing and regulating the vault-wide operations (e.g., configure parameters, assign roles, and execute privileged operations). 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 their related privileged accesses in current contracts.

```
101     function setMinDeposit(uint256 _minDeposit) external onlyRole(DEFAULT_ADMIN_ROLE)
            {...}

103     ...
104     function setCustodian(address _custodian) external onlyRole(DEFAULT_ADMIN_ROLE)
            {...}

106     ...
107     function setStageWindows(uint64 _depositWindowDays, uint64 _processingWindowDays)
            external onlyRole(DEFAULT_ADMIN_ROLE) {...}

109     ...
110     function recoverToken(
111         address token,
112         address to,
113         uint256 amount
114     ) external onlyRole(DEFAULT_ADMIN_ROLE) {...}

116     ...
117     function setWhitelist(address[] calldata accounts, bool[] calldata whitelisted)
            external onlyRole(KEEPER_ROLE) {...}

119     ...
120     function advanceEpoch() external onlyRole(KEEPER_ROLE) {...}
```

Listing 3.3: Example Privileged Operations in `Vault`

We understand the need of the privileged function for contract maintenance, but at the same time the extra power to the privileged account may also be a counter-party risk to the protocol users. It is worrisome if the privileged account is 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, the audited contract makes use of the proxy contract to allow for future upgrades. The upgrade is a privileged operation and the management of the related admin key also falls in this trust issue.

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 with the use of a multi-sig for the privileged account.

4 Conclusion

In this audit, we have analyzed the design and implementation of the AllDeFi protocol, which is a compliant, institutional-grade yield protocol offering professional strategy management and institutional custody within a transparent, KYC-verified framework. The audited Vault facilitates user deposits through an epoch-based structure, with deposits processed in a controlled and secure manner. 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 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.

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