PARAM Token

SMART CONTRACT AUDIT

28.03.2024

Made in Germany by Softstack.io

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Victory Vault Limited. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (26.03.2024)	Layout
0.4 (27.03.2024)	Automated Security Testing Manual Security Testing
0.5 (28.03.2024)	Verify Claims and Test Deployment
0.6 (28.03.2024)	Testing SWC Checks
0.9 (28.03.2024)	Summary and Recommendation
1.0 (28.03.2024)	Final document

2. About the Project and Company

Company address:

Victory Vault Limited
2nd Floor, Ellen L.Skelton Building
Fishers Lane, Road Town, Tortola
British Virgin Islands, VG1110

Website: https://paramlabs.io

LinkedIn: https://www.linkedin.com/company/param-laboratory

Twitter (X): https://twitter.com/ParamLaboratory

Medium: https://medium.com/@paramlabs

2.1 Project Overview

Param Labs is a pioneering company at the intersection of artificial intelligence and blockchain technologies, dedicated to revolutionizing the rapidly expanding gaming industry, which currently exceeds a valuation of US$200 billion. By tackling significant challenges perceived to hinder the sector's long-term growth, Param Labs is establishing new benchmarks for innovation and user engagement.

The core ethos of Param Labs revolves around empowering gamers with digital ownership over their gaming assets and fostering the creation of user-generated value within gaming experiences. The company's ecosystem is meticulously crafted to facilitate this vision, aiming to seamlessly integrate millions of new users into the benefits of Web3 technology while also empowering individual creators and studios.

Param Labs boasts a team of seasoned professionals with backgrounds from esteemed companies such as Activision, EA, and Ubisoft, as well as acclaimed artists Antoni and Marc Tudisco, renowned for their game and character designs. This team is dedicated to developing cutting-edge tools that enable the effortless utilization of emerging technologies, thereby paving the way for the future of the gaming industry.

The company's flagship project, Kiraverse, is a Web3 multiplayer third-person shooter game that epitomizes Param Labs' commitment to digital ownership and user-generated value. Kiraverse provides players with the ability to compete, earn, and trade digital assets like characters and skins, leveraging innovative technology developed by Param Labs.

In addition to game development, Param Labs is actively constructing infrastructure and tools to empower developers in enhancing their users' blockchain-based experiences. The company's Pixel-to-Poly service, for instance, enables users to convert 2D images into 3D playable in-game characters, compatible not only with Kiraverse but also with popular Web2 titles such as Grand Theft Auto V and Fortnite.

Furthermore, Param Labs has a strong focus on esports and has forged partnerships with major esports organizations worldwide to foster growth within the Web3 gaming ecosystem. The company is also preparing for a token launch that will serve as an ecosystem token for its upcoming games and infrastructure, further solidifying its commitment to driving innovation and advancement within the gaming industry.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to softstack to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to softstack describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

5. Metrics

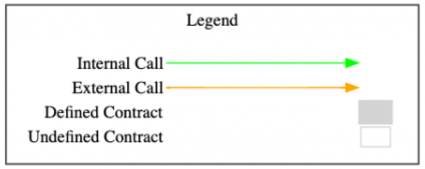
The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

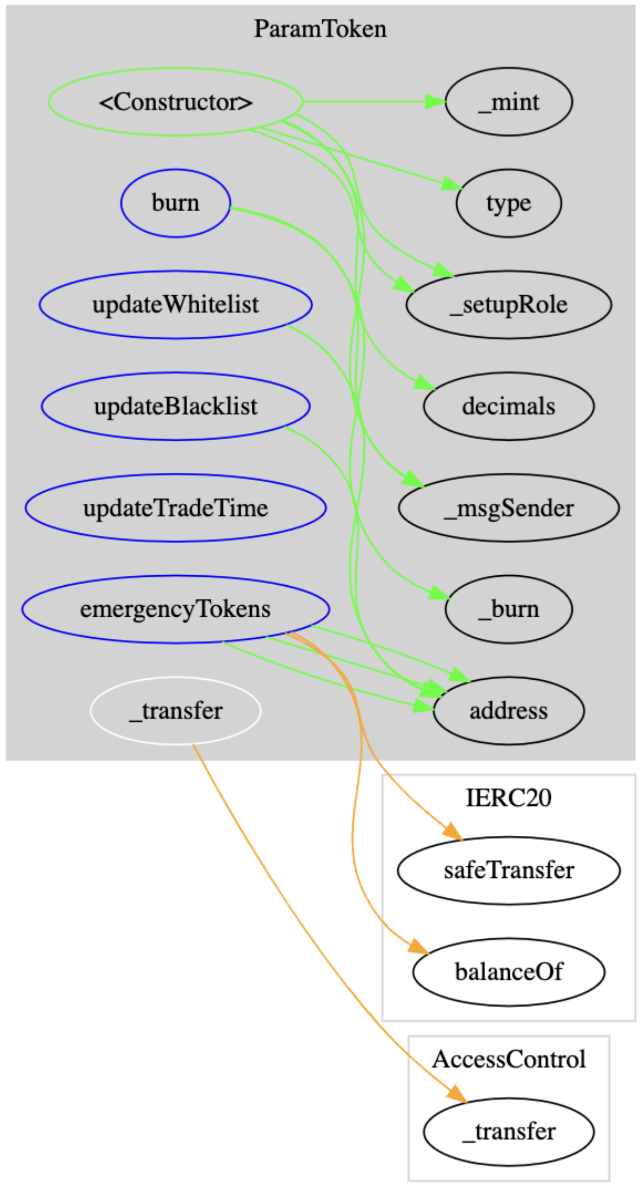
5.1 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

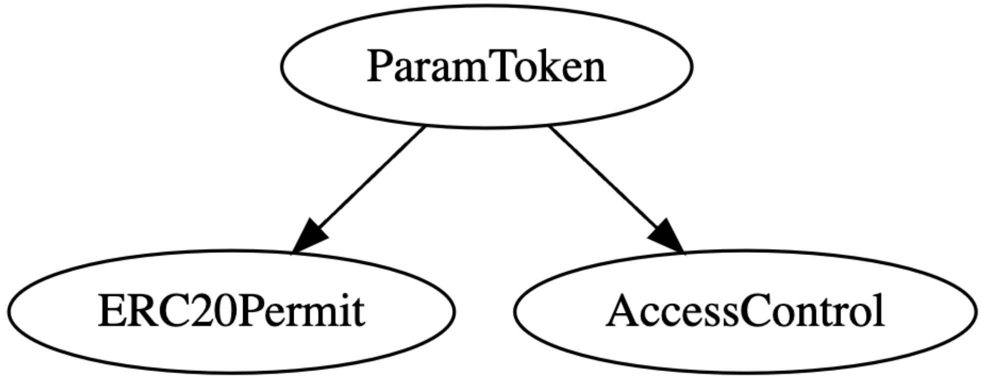
File	Fingerprint (MD5)
./src/active/ParamToken.sol	1ed919c3ce1df4992dbee4f9e877eaa3

5.2 CallGraph

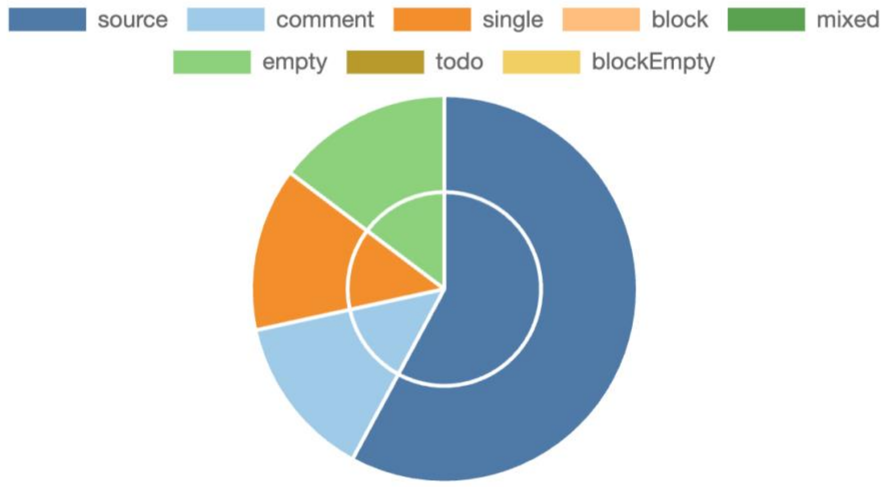


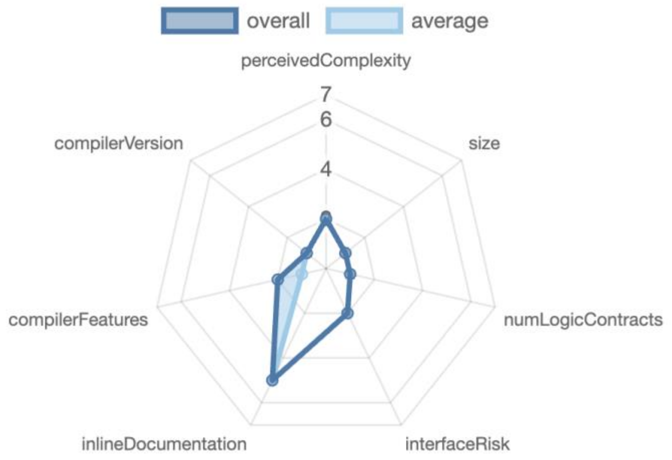


5.3 Inheritance Graph



5.4 Source Lines & Risk





5.5 Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
0.8.8				

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
			yes		

Exposed Functions
This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
5	0

External	Internal	Private	Pure	View
5	8	0	0	0

StateVariables

Total	🌐Public
6	6

5.6 Dependencies / External imports

Dependency / Import Path	Source
@openzeppelin/contracts/access/AccessControl.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.6/contracts/access/AccessControl.sol
@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.6/contracts/token/ERC20/extensions/ERC20Permit.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.6/contracts/token/ERC20/utils/SafeERC20.sol

5.7 Source Unites in Scope

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
src/active/ParamToken.sol	1		82	82	55	13	58
Totals	**1**		**82**	**82**	**55**	**13**	**58**

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

6. Scope of Work

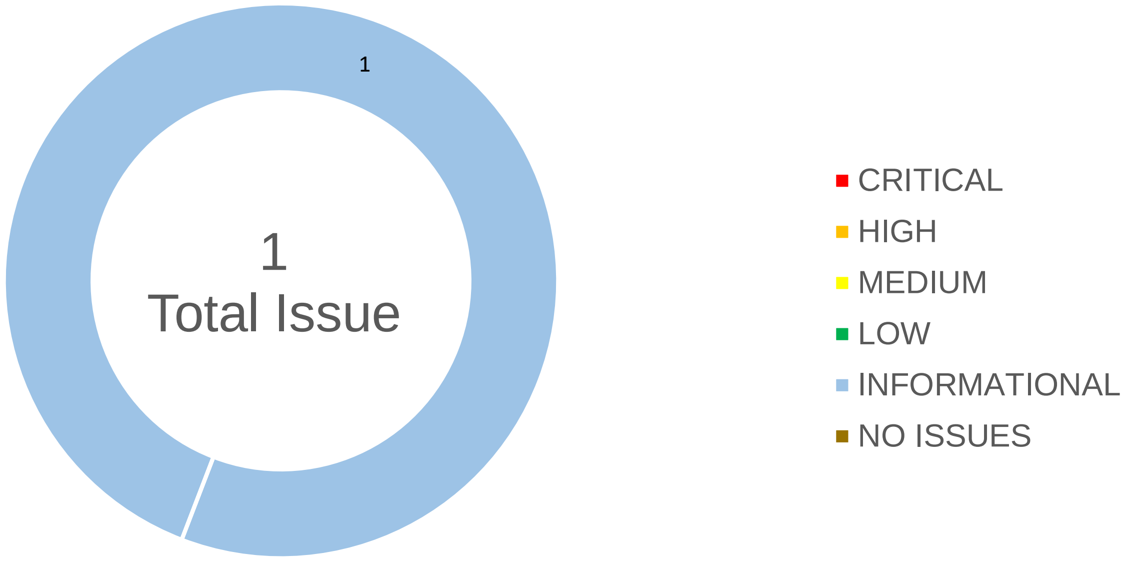
The ParamLabs Team provided us with the files that needs to be tested. The scope of the audit is the PARAM Token contract.

The team put forward the following assumptions regarding the security, usage of the contracts:

1. Compliance with Best Practices: The audit should ensure that the contract adheres to smart contract best practices, including checking for common vulnerabilities such as reentrancy attacks and overflow/underflow issues.
2. Effective Role-Based Access Control: The audit confirms proper assignment and management of roles like OPERATOR and MANAGER, ensuring only authorized entities execute privileged functions.
3. Secure Token Transfer Functions: Token transfer functions are audited to ensure resilience against common vulnerabilities, safeguarding against unauthorized transfers and balance manipulation.
4. Valid Whitelist and Blacklist Mechanisms: The audit validates the functionality of whitelist and blacklist mechanisms, ensuring correct addition/removal of addresses and proper enforcement of transfer restrictions.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Centralized Control of Transfer Function	INFORMATIONAL	ACKNOWLEDGED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, softstack‘s experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, softstack‘s experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, softstack‘s experts found **no Medium issues** in the code of the smart contract.

LOW ISSUES

During the audit, softstack‘s experts found **no Low issues** in the code of the smart contract

INFORMATIONAL ISSUES

During the audit, softstack‘s experts found **one Informational issue** in the code of the smart contract.

6.2.1 Centralized Control of Transfer Function

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: ParamToken.sol

Attack / Description	The contract utilizes whitelist and blacklist mechanisms to restrict token transfers based on specified addresses. However, the ability to add or remove addresses from these lists is centralized and controlled by designated operators. This centralized control introduces a potential informational issue, as it may deviate from the principles of decentralization and transparency often associated with blockchain projects.

	While the contract functions as intended and provides mechanisms for managing token transfers, the reliance on centralized control for managing address lists could impact the perceived level of decentralization and community governance. Projects aiming for greater decentralization may want to consider alternative approaches, such as community-based voting mechanisms or decentralized autonomous organization (DAO) structures, to distribute control and decision-making among token holders.

Code	Line 70 - 81 (ParamToken.sol):
	```
	/// @notice Override transfer function to check blacklist and whitelist also trading time and amount
	  function _transfer(address sender, address recipient, uint256 amount) internal override {
	    /// @dev The blacklist address can't transfer or receive tokens. This is the highest priority
	    require(!blacklist[sender] && !blacklist[recipient], "0x1");

	    /// @dev The whitelist address can transfer or receive tokens without trading restriction
	    if (!whitelist[sender] && !whitelist[recipient]) {
	      require(block.timestamp >= startTime, "0x2");
	      require(amount <= maxAmount, "0x3");
	    }
	    super._transfer(sender, recipient, amount);
	  }
	```

Result/Recommendation	1. Secure the OPERATOR role by implementing a multi-signature wallet mechanism. Utilizing multi-signature wallets for managing critical roles like OPERATOR adds an extra layer of security by requiring multiple parties to authorize transactions or changes. This reduces the risk of unauthorized access or misuse of privileged roles within the contract. 2. Ensure transparent communication of the blacklist and whitelist trading functionality with the public audience. Clearly document the criteria and processes for adding or

	removing addresses from these lists, including the rationale behind such decisions. Providing this information openly to the community fosters trust and understanding, allowing stakeholders to better comprehend how address lists are managed within the ecosystem.

6.3 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✓
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✓
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✓
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✓
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✓
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✓
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✓

ID	Title	Relationships	Test Result
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✓
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✓
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✓
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✓
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✓
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✓
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✓

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✓
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✓
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✓
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✓
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✓
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✓
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✓
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✓

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✓
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✓
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✓
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓

6.4 Verify Claims

6.4.1 Compliance with Best Practices: The audit should ensure that the contract adheres to smart contract best practices, including checking for common vulnerabilities such as reentrancy attacks and overflow/underflow issues.
Status: tested and verified ✅

6.4.2 Effective Role-Based Access Control: The audit confirms proper assignment and management of roles like OPERATOR and MANAGER, ensuring only authorized entities execute privileged functions.
Status: tested and verified ✅

6.4.3 Secure Token Transfer Functions: Token transfer functions are audited to ensure resilience against common vulnerabilities, safeguarding against unauthorized transfers and balance manipulation.
Status: tested and verified ✅

6.4.4 Valid Whitelist and Blacklist Mechanisms: The audit validates the functionality of whitelist and blacklist mechanisms, ensuring correct addition/removal of addresses and proper enforcement of transfer restrictions.
Status: tested and verified ✅

7. Executive Summary

Two independent softstack experts performed an unbiased and isolated audit of the smart contract codebase provided by the ParamLabs team. The main objective of the audit was to verify the security and functionality claims of the smart contract. The audit process involved a thorough manual code review and automated security testing.

Overall, the audit identified a total of one issue, classified as follows:
- No critical issues were found.
- No high severity issues were found.
- No medium severity issues were found.
- No low severity issues were discovered
- One informational issues were identified

The audit report provides detailed descriptions of each identified issue, including severity levels, CWE classifications, and recommendations for mitigation. It also includes code snippets, where applicable, to demonstrate the issues and suggest possible fixes. Based on the nature of the finding and adherence to the business logic, we recommend that the ParamLabs team review the suggestion. However, considering the cleanliness and straightforwardness of the codebase, there may be no immediate need for action.

PARAM Token deployment on Ethereum mainnet:
https://etherscan.io/address/0x69A1e699f562D7Af66Fc6cc473d99f4430C3AcD2#code

8. About the Auditor

Established in 2017 under the name Chainsulting, and rebranded as softstack GmbH in 2023, softstack has been a trusted name in Web3 Security space. Within the rapidly growing Web3 industry, softstack provides a comprehensive range of offerings that include software development, cybersecurity, and consulting services. Softstack's competency extends across the security landscape of prominent blockchains like Solana, Tezos, TON, Ethereum and Polygon. The company is widely recognized for conducting thorough code audits aimed at mitigating risk and promoting transparency.

The firm's proficiency lies particularly in assessing and fortifying smart contracts of leading DeFi projects, a testament to their commitment to maintaining the integrity of these innovative financial platforms. To date, softstack plays a crucial role in safeguarding over $100 billion worth of user funds in various DeFi protocols.

Underpinned by a team of industry veterans possessing robust technical knowledge in the Web3 domain, softstack offers industry-leading smart contract audit services. Committed to evolving with their clients' ever-changing business needs, softstack's approach is as dynamic and innovative as the industry it serves.

Check our website for further information: https://softstack.io

How We Work

1 PREPARATION
Supply our team with audit ready code and additional materials

2 COMMUNICATION
We setup a real-time communication tool of your choice or communicate via e-mails.

3 AUDIT
We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.

4 FIXES
Your development team applies fixes while consulting with our auditors on their safety.

5 REPORT
We check the applied fixes and deliver a full report on all steps done.