EtherAuthority
etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Phala Network
Website: phala.network
Platform: Base Chain Network
Language: Solidity
Date: June 12th, 2024

Table of contents

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Phala Network smart contract from phala.network was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on June 12th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

Website Details

Phala Network is a decentralized cloud computing platform designed for Web3 applications. It leverages secure and scalable computing solutions through Phat Contracts, which enable trustless off-chain computation. This allows developers to create a wide range of Web3 use cases, including secure data processing and AI integrations.

Key features of Phala Network include:

1. Phat Contracts: Smart contracts enable developers to securely and trustlessly perform off-chain computation using both no-code and native code toolkits.
2. Compute to Earn: Users can provide hardware to the network and earn rewards in the form of PHA tokens.
3. Staking and Governance: Users can stake PHA tokens to help secure the network and participate in governance to influence the direction of the network's development.
4. SubBridge: This feature enables the transfer of tokens between the Phala Blockchain and other blockchains, enhancing interoperability within the Polkadot ecosystem.

5. Community Engagement: Phala Network has incorporated AwesomeQA, an AI-powered chatbot, to enhance community support on their Discord server and website, providing 24/7 multilingual assistance and accurate responses.

Code Details

- The `TokenImplementation` contract is an advanced ERC20 token implementation with several additional features.
- It is based on the OpenZeppelin ERC20 implementation and includes support for EIP-712 (typed structured data hashing and signing) and EIP-2612 (permit function to approve spending via signatures). The contract also uses the Beacon Proxy pattern for upgradeability and includes custom initialization logic.
- This contract is suitable for creating a customized ERC20 token with advanced features like off-chain approvals and metadata updates, managed by the token owner.

Audit scope

Name	**Code Review and Security Analysis Report for Phala Network Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	TokenImplementation.sol
Smart Contract Code	0x5537857664b0f9efe38c9f320f75fef23234d904
Audit Date	June 12th,2024
Audit Result	**Passed**

Code Audit History

0 Total Findings	0 Critical	0 High	0 Medium	0 Low	0 Informational

Severity Definitions

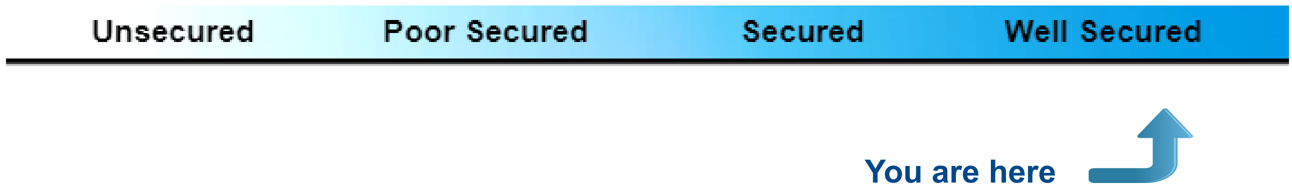
0	**Critical**	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0	**High**	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0	**Medium**	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
0	**Low**	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
0	**Lowest / Informational / Best Practice**	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Ownership Control: ● Unlimited token minting. ● The Owner can burn anyone's token. ● The current owner can transfer the ownership. ● The owner can renounce ownership.	**YES, This is valid. We advise renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Well Secured"**.Also, these contracts contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 0 very low-level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	No
Fee Check	Not Detected
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	No
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	Yes
Is it a Proxy Contract?	Yes
Is it used Open Source?	Yes
External Call Risk?	No
Balance Modifiable?	No
Can Take Ownership?	Yes
Ownership Renounce?	Yes
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Phala Network are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Phala Network.

The EtherAuthority team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Phala Network smart contract code in the form of a basescan web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

TokenImplementation.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initialize	write	initializer	No Issue
3	_initializeNativeToken	internal	Passed	No Issue
4	_initializePermitStateIfNeeded	internal	Passed	No Issue
5	name	read	Passed	No Issue
6	symbol	read	Passed	No Issue
7	owner	read	Passed	No Issue
8	decimals	read	Passed	No Issue
9	totalSupply	read	Passed	No Issue
10	chainId	read	Passed	No Issue
11	nativeContract	read	Passed	No Issue
12	balanceOf	read	Passed	No Issue
13	transfer	write	Passed	No Issue
14	allowance	read	Passed	No Issue
15	approve	write	Passed	No Issue
16	transferFrom	write	Passed	No Issue
17	increaseAllowance	write	Passed	No Issue
18	decreaseAllowance	write	Passed	No Issue
19	_transfer	internal	Passed	No Issue
20	mint	write	access only Owner	No Issue
21	_mint	internal	Passed	No Issue
22	burn	write	access only Owner	No Issue
23	_burn	internal	Passed	No Issue
24	_approve	internal	Passed	No Issue
25	updateDetails	write	access only Owner	No Issue
26	onlyOwner	modifier	Passed	No Issue
27	initializer	modifier	Passed	No Issue
28	_domainSeparatorV4	internal	Passed	No Issue
29	_buildDomainSeparator	internal	Passed	No Issue
30	_hashTypedDataV4	internal	Passed	No Issue
31	permit	write	Passed	No Issue
32	DOMAIN_SEPARATOR	read	Passed	No Issue
33	eip712Domain	read	Passed	No Issue
34	_eip712DomainVersion	internal	Passed	No Issue
35	_eip712DomainNameHashed	internal	Passed	No Issue
36	_eip712DomainSalt	internal	Passed	No Issue
37	nonces	read	Passed	No Issue
38	_useNonce	internal	Passed	No Issue

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

No Low-severity vulnerabilities were found.

Very Low / Informational / Best practices:

No Very-Low-severity vulnerabilities were found.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble.

The following are Admin functions:

TokenImplementation.sol

- mint: The owner can mint new tokens.
- burn: The owner can burn tokens.
- updateDetails: The owner can update details.

Ownable.sol

- transferOwnership: Allows the current owner to transfer control of the contract to a newOwner.
- renounceOwnership: Renounce the ownership of the contract to leave the contract without an owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a basescan web link. And we have used all possible tests based on given objects as files. We observed no issue in the smart contracts. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Well Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

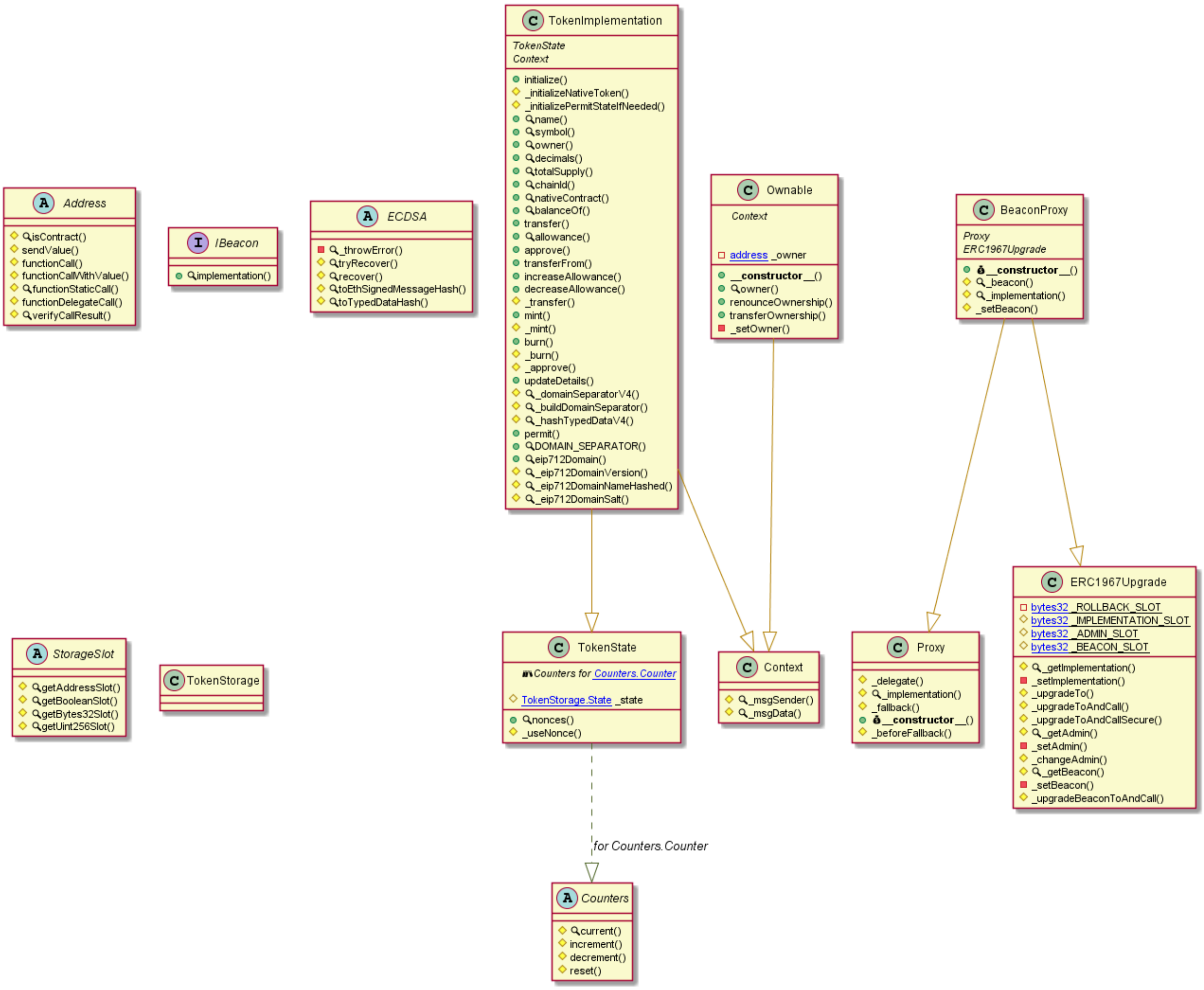
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Phala Network



Email: audit@EtherAuthority.io

Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

TokenImplementation.sol

```
INFO:Detectors:
TokenImplementation.permit(address,address,uint256,uint256,uint8,bytes32,bytes32)
(TokenImplementation.sol#1216-1252) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp <= deadline_,ERC20Permit: expired deadline)
(TokenImplementation.sol#1230)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Pragma version^0.8.0 (TokenImplementation.sol#3) allows old versions
solc-0.8.25 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.functionDelegateCall(address,bytes,string)
(TokenImplementation.sol#176-185):
	- (success,returndata) = target.delegatecall(data) (TokenImplementation.sol#183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function TokenImplementation.DOMAIN_SEPARATOR() (TokenImplementation.sol#1258-1260)
is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Slither:TokenImplementation.sol analyzed (13 contracts with 93 detectors), 50 result(s)
found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

TokenImplementation.sol

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 347:16:

Gas costs:
Gas requirement of function TokenImplementation.burn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 232:4:

Gas costs:
Gas requirement of function TokenImplementation.mint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 220:4:

Similar variable names:
TokenImplementation.mint(address,uint256) : Variables have very similar names "account_" and "amount_". Note: Modifiers are currently not considered by this static analysis.
Pos: 221:14:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 209:8:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

TokenImplementation.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:3
global import of path @openzeppelin/contracts/access/Ownable.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:5
global import of path @openzeppelin/contracts/utils/Context.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:6
global import of path @openzeppelin/contracts/proxy/beacon/BeaconProxy.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:7
global import of path @openzeppelin/contracts/utils/cryptography/ECDSA.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:8
global import of path @openzeppelin/contracts/utils/Counters.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)
Pos: 1:10
Explicitly mark visibility of state
Pos: 5:51
Visibility modifier must be first in list of modifiers
Pos: 19:83
Error message for require is too long
Pos: 9:188
Error message for require is too long
Pos: 9:201
Error message for require is too long
Pos: 9:208
Error message for require is too long
Pos: 9:209
Error message for require is too long
Pos: 9:212
Error message for require is too long
Pos: 9:236
```

```
Error message for require is too long
Pos: 9:239
Error message for require is too long
Pos: 9:247
Error message for require is too long
Pos: 9:248
Code contains empty blocks
Pos: 83:325
Avoid making time-based decisions in your business logic
Pos: 17:346
```

Software analysis result:

This software reported many false positive results and some were informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io