Coinsult

Request your audit at coinsult.net

# Advanced Manual
# **Smart Contract Audit**

August 6, 2024

CoinsultAudits

t.me/coinsult_tg

coinsult.net

Audit requested by

Hyperlayer

0x727c46a924de53dd252c9e2693082a120326e484

# Global Overview

## Manual Code Review

In this audit report we will highlight the following issues:

| Vulnerability Level | Total | Pending | Acknowledged | Resolved |
|---|---|---|---|---|
| Informational | 0 | 0 | 0 | 0 |
| Low-Risk | 4 | 4 | 0 | 0 |
| Medium-Risk | 0 | 0 | 0 | 0 |
| High-Risk | 0 | 0 | 0 | 0 |

## Centralization Risks

Coinsult checked the following privileges:

| Contract Privilege | Description |
|---|---|
| Owner needs to enable trading? | Owner needs to manually enable trading |
| Owner can mint? | Owner cannot mint new tokens |
| Owner can blacklist? | Owner cannot blacklist addresses |
| Owner can set fees? | Owner can set the sell fee to 0% |
| Owner can exclude from fees? | Owner can exclude from fees |
| Can be honeypotted? | Owner cannot pause the contract |
| Owner can set Max TX amount? | Owner cannot set max transaction amount |

More owner priviliges are listed later in the report.

# Table of Contents

# Audit Summary

| | |
|---|---|
| **Project Name** | Hyperlayer |
| **Website** | https://hyperlayer.pro |
| **Blockchain** | Ethereum |
| **Smart Contract Language** | Solidity |
| **Contract Address** | 0x727c46a924de53dd252c9e2693082a120326e484 |
| **Audit Method** | Static Analysis, Manual Review |
| **Date of Audit** | 6 August 2024 |

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

# Audit Scope

Coinsult was comissioned by Hyperlayer to perform an audit based on the following code:

**https://etherscan.io/token/0x727c46a924de53dd252c9e2693082a120326e484#code**

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

## Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

### Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

### Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

### Used tools

- Slither: Solidity static analysis framework
- Remix: IDE Developer Tool
- CWE: Common Weakness Enumeration
- SWC: Smart Contract Weakness Classification and Test Cases
- DEX: Testnet Blockchains

# Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

| Vulnerability Level | Description |
| --- | --- |
| Informational | Does not compromise the functionality of the contract in any way |
| Low-Risk | Won't cause any problems, but can be adjusted for improvement |
| Medium-Risk | Will likely cause problems and it is recommended to adjust |
| High-Risk | Will definitely cause problems, this needs to be adjusted |

Coinsult has four statuses that are used for each risk level. Below we explain them briefly.

| Risk Status | Description |
| --- | --- |
| Total | Total amount of issues within this category |
| Pending | Risks that have yet to be addressed by the team |
| Acknowledged | The team is aware of the risks but does not resolve them |
| Resolved | The team has resolved and remedied the risk |

# SWC Attack Analysis

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

| ID | Description | Status |
|---|---|---|
| SWC-100 | Function Default Visibility | Passed |
| SWC-101 | Integer Overflow and Underflow | Passed |
| SWC-102 | Outdated Compiler Version | Passed |
| SWC-103 | Floating Pragma | Failed |
| SWC-104 | Unchecked Call Return Value | Passed |
| SWC-105 | Unprotected Ether Withdrawal | Passed |
| SWC-106 | Unprotected SELFDESTRUCT Instruction | Passed |
| SWC-107 | Reentrancy | Passed |
| SWC-108 | State Variable Default Visibility | Passed |
| SWC-109 | Uninitialized Storage Pointer | Passed |
| SWC-110 | Assert Violation | Passed |
| SWC-111 | Use of Deprecated Solidity Functions | Passed |
| SWC-112 | Delegatecall to Untrusted Callee | Passed |
| SWC-113 | DoS with Failed Call | Passed |
| SWC-114 | Transaction Order Dependence | Passed |
| SWC-115 | Authorization through tx.origin | Passed |

| | | |
|---|---|---|
| SWC-116 | Block values as a proxy for time | Passed |
| SWC-117 | Signature Malleability | Passed |
| SWC-118 | Incorrect Constructor Name | Passed |
| SWC-119 | Shadowing State Variables | Passed |
| SWC-120 | Weak Sources of Randomness from Chain Attributes | Passed |
| SWC-121 | Missing Protection against Signature Replay Attacks | Passed |
| SWC-122 | Lack of Proper Signature Verification | Passed |
| SWC-123 | Requirement Violation | Passed |
| SWC-124 | Write to Arbitrary Storage Location | Passed |
| SWC-125 | Incorrect Inheritance Order | Passed |
| SWC-126 | Insufficient Gas Griefing | Passed |
| SWC-127 | Arbitrary Jump with Function Type Variable | Passed |
| SWC-128 | DoS With Block Gas Limit | Passed |
| SWC-129 | Typographical Error | Passed |
| SWC-130 | Right-To-Left-Override control character (U+202E) | Passed |
| SWC-131 | Presence of unused variables | Passed |
| SWC-132 | Unexpected Ether balance | Passed |
| SWC-133 | Hash Collisions With Multiple Variable Length Arguments | Passed |
| SWC-134 | Message call with hardcoded gas amount | Passed |
| SWC-135 | Code With No Effects | Passed |
| SWC-136 | Unencrypted Private Data On-Chain | Passed |

| Error Code | Description |
| --- | --- |
| SLT: 078 | Conformance to numeric notation best practices |

**Low-Risk:** Could be fixed, will not bring problems.

### Too many digits

Literals with many digits are difficult to read and review.

```
uint256 private  mintAmount = 100000 * 10**uint256(decimals());
```

### Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

### Exploit scenario

```
contract MyContract{
    uint 1_ether = 1000000000000000000;
}
```

While `1_ether` looks like `1 ether`, it is `10 ether`. As a result, it's likely to be used incorrectly.

| Error Code | Description |
| --- | --- |
| SWC: 103 | Floating Pragma |

**Low-Risk:** Could be fixed, will not bring problems.

### Floating Pragma

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

```
pragma solidity ^0.8.0;
```

### Recommendation

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

| Error Code | Description |
| --- | --- |
| SLT: 054 | Missing Events Arithmetic |

**Low-Risk:** Could be fixed, will not bring problems.

## Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function endedPublicSaleEnabled() external onlyOwner{
    require(!publicSaleEnabled, "Public sale has ended");
    publicSaleEnabled = true;
}
```

## Recommendation

Emit an event for critical parameter changes.

## Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

`updateOwner()` has no event, so it is difficult to track off-chain changes in the buy price.

| Error Code | Description |
| --- | --- |
| CS: 071 | Using safemath in Solidity 0.8.0+ |

**Low-Risk:** Could be fixed, will not bring problems.

## Using safemath in Solidity 0.8.0+

SafeMath is generally not needed starting with Solidity 0.8, since the compiler now has built in overflow checking.

```
library SafeMath {
/**
 * @dev Returns the addition of two unsigned integers, with an overflow flag.
 *
 * _Available since v3.4._
 */
function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }
}

/**
 * @dev Returns the substraction of two unsigned integers, with an overflow flag.
```

## Recommendation

Check if you really need SafeMath and consider removing it.

| Error Code | Description |
| --- | --- |
| CS: 016 | Initial Supply |

**Low-Risk:** Could be fixed, will not bring problems.

## Initial Supply

When the contract is deployed, the contract deployer receives all of the initially created assets. Since the deployer and/or contract owner can distribute tokens without consulting the community, this could be a problem.

## Recommendation

Private keys belonging to the employer and/or contract owner should be stored properly. The initial asset allocation procedure should involve consultation with the community.

# Maximum Fee Limit Check

| Error Code | Description |
| --- | --- |
| CEN-01 | Centralization: Operator Fee Manipulation |

Coinsult tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.

| Type of fee | Description |
| --- | --- |
| Max transfer fee | 0% |
| Max buy fee | 0% |
| Max sell fee | 0% |

# Contract Honeypot Check

| Error Code | Description |
| --- | --- |
| CEN-02 | Centralization: Operator Pausability |

Coinsult tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

| Privilege Check | Description |
| --- | --- |
| Can owner pause the contract? | Owner cannot pause the contract |

# Max Transaction Amount Check

| Error Code | Description |
| --- | --- |
| CEN-03 | Centralization: Operator Transaction Manipulation |

Coinsult tests if the owner of the smart contract can set the maximum amount of a transaction. If the transaction exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

| Privilege Check | Description |
| --- | --- |
| Can owner set max tx amount? | Owner cannot set max transaction amount |

# Exclude From Fees Check

| Error Code | Description |
| --- | --- |
| CEN-04 | Centralization: Operator Exclusion |

Coinsult tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.

| Privilege Check | Description |
| --- | --- |
| Can owner exclude from fees? | Owner can exclude from fees |

## Function

```
function setExcludedFromFees(address account, bool state) external onlyOwner{
    require(_isExcludedFromFees[account] != state, "Value already set");
    _isExcludedFromFees[account] = state;
}
```

# Ability To Mint Check

| Error Code | Description |
| --- | --- |
| CEN-05 | Centralization: Operator Increase Supply |

Coinsult tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token's total supply as non-fixed, allowing the token owner to "mint" more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A method to disable minting can also be added to stop the minting process irreversibly.

Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

| Privilege Check | Description |
| --- | --- |
| Can owner mint? | Owner cannot mint new tokens |

# Enable Trading

| Error Code | Description |
| --- | --- |
| CEN-06 | Centralization: Operator enable trading |

Coinsult tests if the owner of the smart contract needs to manually enable trading before everyone can buy & sell. If the owner needs to manually enable trading, this poses a high centralization risk.

If the owner needs to manually enable trading, make sure to check if the project has a SAFU badge or a trusted KYC badge. Always DYOR when investing in a project that needs to manually enable trading.

| Privilege Check | Description |
| --- | --- |
| Owner needs to enable trading? | Owner needs to manually enable trading |

# Ability To Blacklist Check

| Error Code | Description |
| --- | --- |
| CEN-07 | Centralization: Operator Dissalows Wallets |

Coinsult tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting methods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This method can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

| Privilege Check | Description |
| --- | --- |
| Can owner blacklist? | Owner cannot blacklist addresses |

# Other Owner Privileges Check

| Error Code | Description |
| --- | --- |
| CEN-100 | Centralization: Operator Priviliges |

Coinsult lists all important contract methods which the owner can interact with.

Owner can withdraw tokens from the contract address

# Notes

## Notes by Hyperlayer

No notes provided by the team.

## Notes by Coinsult

No notes provided by Coinsult

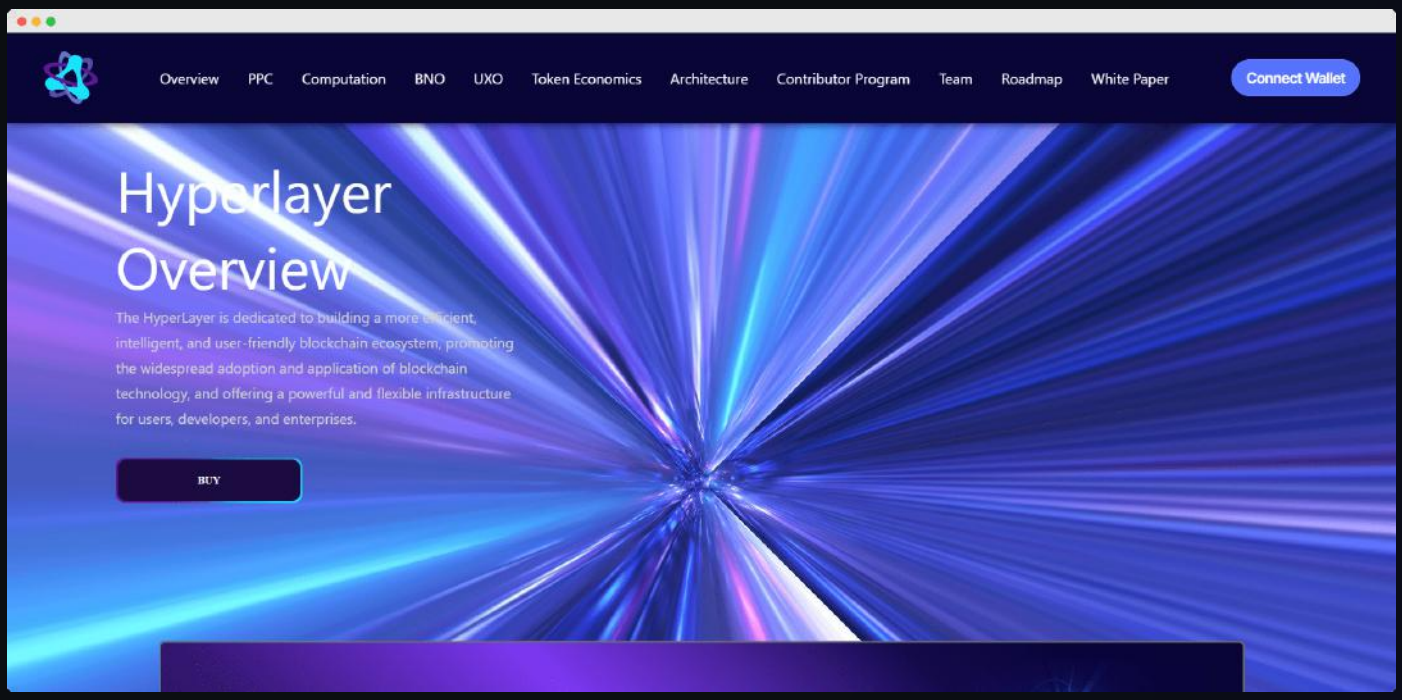
# Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
contract Hyperlayer is ERC20, Ownable {
using SafeMath for uint256;
using Address for address payable;
uint256 private  mintAmount = 25000 * 10**uint256(decimals());
uint256 private constant mintETHAmount = 0.05 ether;
mapping (address =&gt; bool) private _isExcludedFromFees;
mapping(address =&gt; bool) public liquidityPools;
bool public publicSaleEnabled = false;
```

# Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.




| Type of check | Description |
| --- | --- |
| Mobile friendly? | The website is mobile friendly |
| Contains jQuery errors? | The website does not contain jQuery errors |
| Is SSL secured? | The website is SSL secured |
| Contains spelling errors? | The website does not contain spelling errors |

# Certificate of Proof

Not KYC verified by Coinsult

## Hyperlayer

**Audited by Coinsult.net**

**Date: 6 August 2024**

✔ Advanced Manual Smart Contract Audit

# Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Coinsult

coinsult.net

# End of report
# **Smart Contract Audit**


CoinsultAudits

info@coinsult.net

coinsult.net