hashlock.

Security Audit

Levva (Token)

January 2025 info@hashlock.com.au Hashlock Pty Ltd

Table of Contents

CAUTION

THIS DOCUMENT IS A SECURITY AUDIT REPORT AND MAY CONTAIN CONFIDENTIAL INFORMATION. THIS INCLUDES IDENTIFIED VULNERABILITIES AND MALICIOUS CODE WHICH COULD BE USED TO COMPROMISE THE PROJECT. THIS DOCUMENT SHOULD ONLY BE FOR INTERNAL USE UNTIL ISSUES ARE RESOLVED. ONCE VULNERABILITIES ARE REMEDIATED, THIS REPORT CAN BE MADE PUBLIC. THE CONTENT OF THIS REPORT IS OWNED BY HASHLOCK PTY LTD FOR USE OF THE CLIENT.

Executive Summary

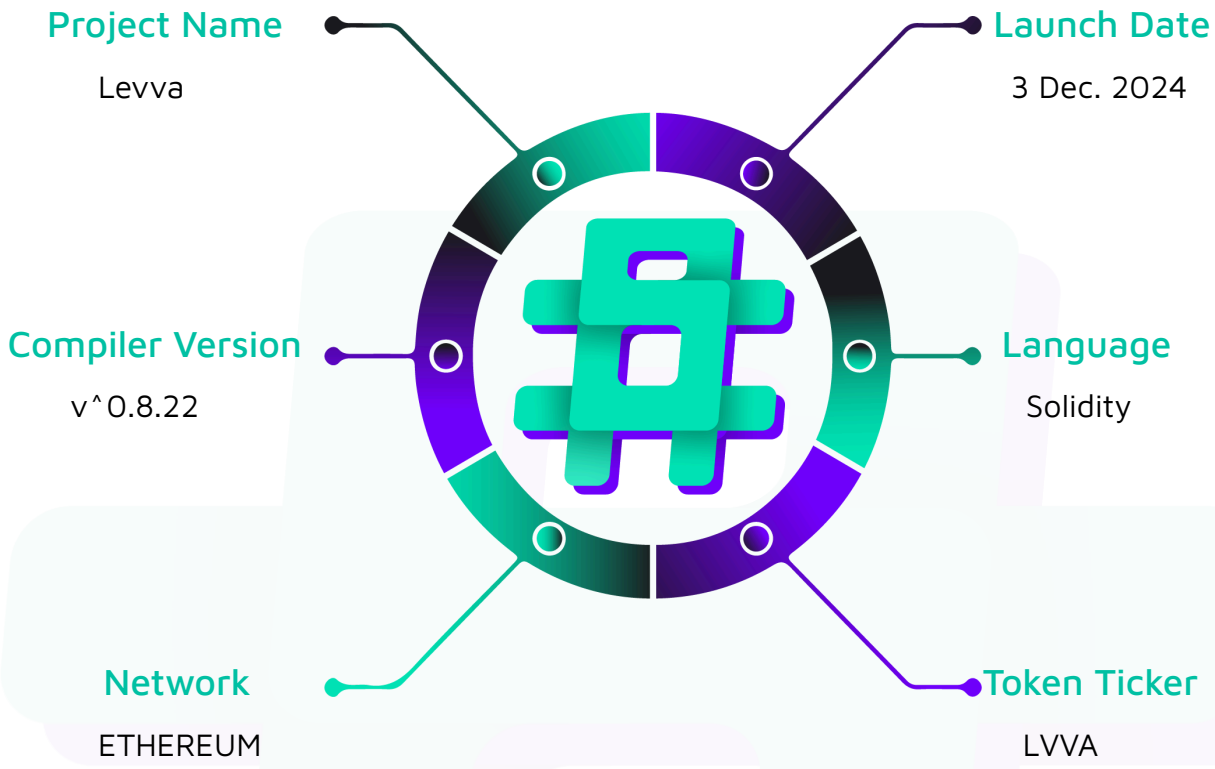
The Levva team partnered with Hashlock to conduct a security audit of their LevvaToken.sol smart contract. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts are secure.

Project Context

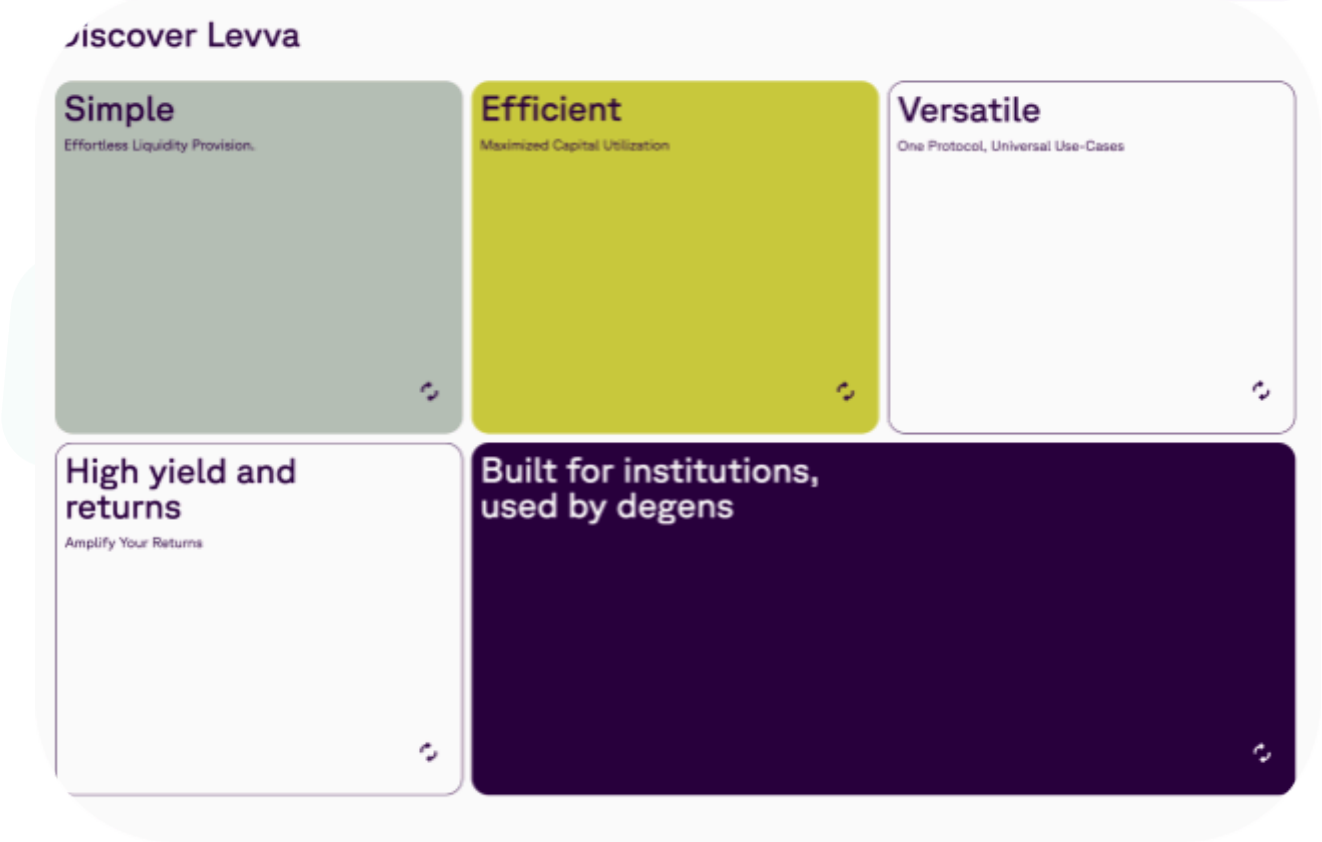
Levva is a decentralized finance (DeFi) platform leveraging Web3 technologies to offer trustless financial services like lending, borrowing, staking, and yield farming. By utilizing smart contracts on a decentralized network, it ensures secure, permissionless transactions and user control through non-custodial Web3 wallets like MetaMask. The platform likely integrates with other DeFi protocols, enhancing its composability and optimizing yields for users. Governance tokens may empower the community to shape the platform's future, while strong security measures, such as audits, help mitigate risks. Levva.fi embodies decentralization, promoting financial autonomy and community-driven growth.

Project Name: Levva
Project Type: Token
Compiler Version: ^0.8.22
Website: www.levva.fi
Logo:

Visualised Context:



Project Visuals:



Audit scope

We at Hashlock audited the solidity code within the Levva project, the scope of work included a comprehensive review of the smart contracts listed below. We tested the smart contracts to check for their security and efficiency. These tests were undertaken primarily through manual line-by-line analysis and were supported by software-assisted testing.

Description	**Levva Protocol Smart Contracts**
Platform	**Ethereum / Solidity**
Audit Date	**January, 2025**
Contract 1	LevvaToken.sol
Contract 1 MD5 Hash	e5d6dd27d3038f5e4923f3cf68324b04
GitHub Commit Hash	56f9c15d0c62a3cc70b862ff897a8de9ebe7a4d2

Security Rating

After Hashlock's Audit, we found the smart contracts to be **"Secure"**. The contracts all follow simple logic, with correct and detailed ordering. They use a series of interfaces, and the protocol uses a list of Open Zeppelin contracts. We initially identified some significant vulnerabilities that have since been addressed.

The 'Hashlocked' rating is reserved for projects that ensure ongoing security via bug bounty programs or on chain monitoring technology.

All issues uncovered during automated and manual analysis were meticulously reviewed and applicable vulnerabilities are presented in the Audit Findings section. The general security overview is presented in the Standardised Checks section and the project's contract functionality is presented in the Intended Smart Contract Functions section.

All vulnerabilities initially identified have now been resolved and acknowledged.

Hashlock found:

1 Medium severity vulnerabilities

3 Low severity vulnerabilities

1 Gas Optimisations

Caution: *Hashlock's audits do not guarantee a project's success or ethics, and are not liable or responsible for security. Always conduct independent research about any project before interacting.*

Intended Smart Contract Functions

Claimed Behaviour	Actual Behaviour
LevvaToken.sol - ERC20 token with 18 decimals - Max supply of 2 billion tokens - Mints 1.25 billion tokens to 4 different addresses in the constructor - Allows admin to: - Mint new tokens	**Contract achieves this functionality.**

Code Quality

This audit scope involves the smart contracts of the Levva project, as outlined in the Audit Scope section. All contracts, libraries, and interfaces mostly follow standard best practices and to help avoid unnecessary complexity that increases the likelihood of exploitation, however, some refactoring was required.

The code is very well commented on and closely follows best practice nat-spec styling. All comments are correctly aligned with code functionality.

Audit Resources

We were given the Levva project smart contract code in the form of Github access.

As mentioned above, code parts are well commented. The logic is straightforward, and therefore it is easy to quickly comprehend the programming flow as well as the complex code logic. The comments are helpful in providing an understanding of the protocol's overall architecture.

Dependencies

As per our observation, the libraries used in this smart contracts infrastructure are based on well-known industry standard open source projects.
Apart from libraries, its functions are used in external smart contract calls.

Severity Definitions

The severity levels assigned to findings represent a comprehensive evaluation of both their potential impact and the likelihood of occurrence within the system. These categorizations are established based on Hashlock's professional standards and expertise, incorporating both industry best practices and our discretion as security auditors. This ensures a tailored assessment that reflects the specific context and risk profile of each finding.

Significance	Description
High	High-severity vulnerabilities can result in loss of funds, asset loss, access denial, and other critical issues that will result in the direct loss of funds and control by the owners and community.
Medium	Medium-level difficulties should be solved before deployment, but won't result in loss of funds.
Low	Low-level vulnerabilities are areas that lack best practices that may cause small complications in the future.
Gas	Gas Optimisations, issues, and inefficiencies
QA	Quality Assurance (QA) findings are informational and don't impact functionality. Supports clients improve the clarity, maintainability, or overall structure of the code.

Status Definitions

Each identified security finding is assigned a status that reflects its current stage of remediation or acknowledgment. The status provides clarity on the handling of the issue and ensures transparency in the auditing process. The statuses are as follows:

Significance	Description
Resolved	The identified vulnerability has been fully mitigated either through the implementation of the recommended solution proposed by Hashlock or through an alternative client-provided solution that demonstrably addresses the issue
Acknowledged	The client has formally recognized the vulnerability but has chosen not to address it due to the high cost or complexity of remediation. This status is acceptable for medium and low-severity findings after internal review and agreement. However, all high-severity findings must be resolved without exception.
Unresolved	The finding remains neither remediated nor formally acknowledged by the client, leaving the vulnerability unaddressed.

Audit Findings

Medium

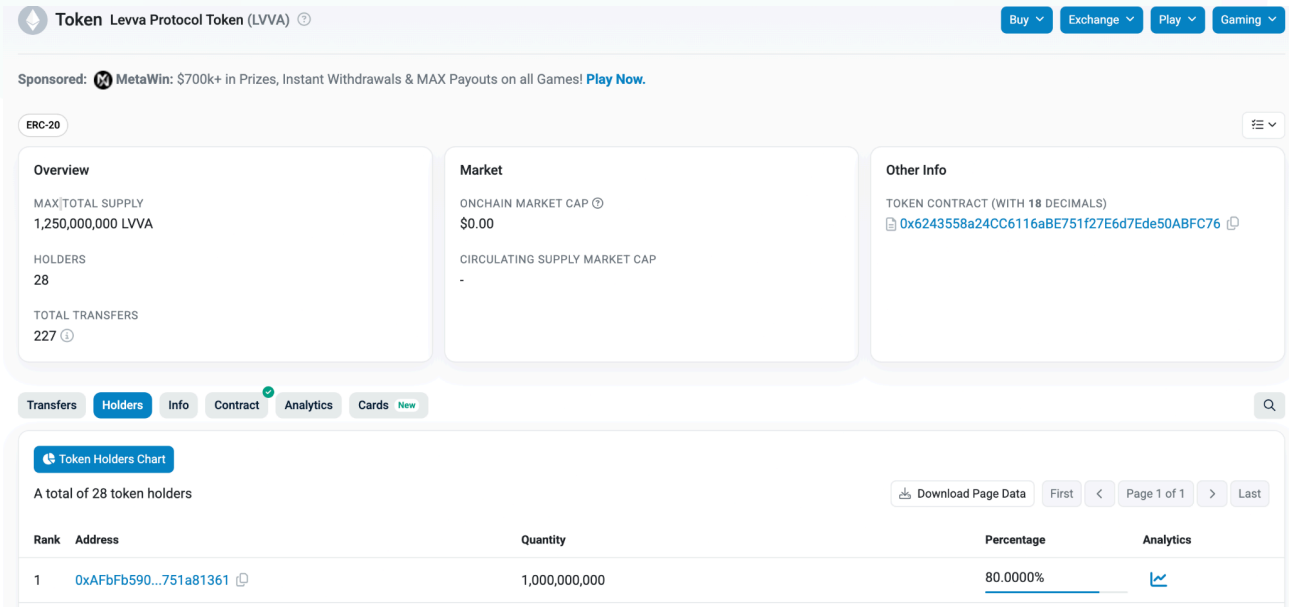
[M-01] LevvaToken - Centralization risk due to unlimited minting power up to maximum total supply and half the maximum total supply being owned by a single EOA

Description

As the token is already deployed, at the time of the audit, 1.25 billion tokens are minted, of which 1 billion (80% of the tokens) is held by a single EOA. This introduces a centralization risk and a single point of failure as most of the currently minted tokens are held by a single EOA. Additionally, the owner address, at the time of the audit, is able to mint 750 million tokens to any address they desire to which is a significant portion of the total available token supply.

Vulnerability Details

Taking a look at the deployed token at Etherscan, it is observed that 80% of the currently minted tokens are held by a single EOA address 0xAFbFb590D65d7E8E15532217e59A48A751a81361.



Additionally, taking a look at the `mint` function below,

```
uint256 public constant MAX_SUPPLY = 2_000_000_000e18;
function mint(address to, uint256 amount) public onlyOwner {
  require(totalSupply() + amount <= MAX_SUPPLY, MaxSupplyExceeded());
  _mint(to, amount);
}
```

It is observed that the owner of the contract has unlimited minting power up to the maximum supply of the token which is 2 billion. At the time of the audit, this amount is 750 million tokens that the owner can mint to any address they desire.

These problems introduce a risk where a single EOA has control over most of the currently minted tokens and the contract owner can mint another significant portion of the maximum total supply to any address they wish.

Impact

The centralization risk introduces single points of failures and causes certain addresses to have significant control over most of the token's supply.

Recommendation

Consider transferring these tokens to a multi-signature wallet that is controlled by a decentralized entity. Additionally, make sure that the contract owner is a multi-signature wallet that is controlled by a decentralized entity.

Note

Levva team informed contract ownership has been transferred to the Gnosis Safe which has 4 owners and a threshold of 3 at this address: `0x32764Ce6edBb6BF39A824cc95246375067c4573e` 1 billion Levva tokens have been transferred to the Gnosis safe that has 4 owners and a threshold of 3 at this address: `0xD20092A19e0488E1283E488e11583B43ba7EA849`

Status

Resolved

Low

[L-01] LevvaToken#constructor - No checks to make sure minted amount does not surpass the maximum supply cap

Description

The constructor mints 1.25 billion tokens to 4 addresses. However, the constructor does not have any implemented checks to make sure that the minted amount does not exceed the maximum supply cap. Even though in this case it does not, it is good practice to have it as a sanity check.

Recommendation

Check if the minted amount in the constructor exceeds the maximum supply cap.

Status

Acknowledged

[L-02] LevvaToken - Use of floating pragma

Description

Contracts use a pragma statement that does not specify a fixed compiler version but instead allows the contract to be compiled with any compatible compiler version. This can lead to various compatibility and stability issues.

Recommendation

Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs for the compiler version that is chosen.

Status

Acknowledged

[L-03] LevvaToken - Use `Ownable2Step` instead of `Ownable`

Description

The currently used OpenZeppelin `Ownable` implementation has a shortcoming that it allows the owner to transfer ownership to a non-existent or mistyped address.

`Ownable2Step` is safer than `Ownable` for smart contracts because the owner cannot accidentally transfer smart contract ownership to a mistyped address. Rather than directly transferring to the new owner, the transfer only completes when the new owner accepts ownership.

Recommendation

Consider using OpenZeppelin's `Ownable2Step` instead of `Ownable`.

Status

Acknowledged

Gas

[G-01] LevvaToken#mint - Function can revert earlier on `address(0)` inputs

Description

Although the `_mint` function in OpenZeppelins ERC20 reverts on `address(0)` inputs, changes can be made to the `mint` function so that it reverts earlier on accidental inputs to save gas.

Recommendation

Change the function as shown below.

```
function mint(address to, uint256 amount) public onlyOwner {
    require(to != address(0), InvalidReceiver());
    require(totalSupply() + amount <= MAX_SUPPLY, MaxSupplyExceeded());
    _mint(to, amount);
}
```

Status

Acknowledged

Centralisation

The Levva project values security and utility over decentralisation.

The Levva project values security and efficiency over decentralisation. Template emphasizes a security-focused approach while ensuring operational functionality. By centralizing critical functions, the project enhances reliability and rapid decision-making while maintaining accountability, ensuring the system remains robust and efficient.

Conclusion

After Hashlock's analysis, the Levva project seems to have a sound and well-tested code base, now that our vulnerability findings have been resolved and acknowledged. Overall, most of the code is correctly ordered and follows industry best practices. The code is well commented on as well. To the best of our ability, Hashlock is not able to identify any further vulnerabilities.

Our Methodology

Hashlock strives to maintain a transparent working process and to make our audits a collaborative effort. The objective of our security audits is to improve the quality of systems and upcoming projects we review and to aim for sufficient remediation to help protect users and project leaders. Below is the methodology we use in our security audit process.

Manual Code Review:

In manually analysing all of the code, we seek to find 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 behaviour when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our methodologies include manual code analysis, user interface interaction, and white box penetration testing. We consider the project's website, specifications, and whitepaper (if available) to attain a high-level understanding of what functionality the smart contract under review contains. We then communicate with the developers and founders to gain insight into their vision for the project. We install and deploy 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 undergo a robust, transparent process for analysing potential security vulnerabilities and seeing them through to successful remediation. When 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 vast 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, and 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 analyse 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 scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the contract details are made public.

Disclaimers

Hashlock's Disclaimer

Hashlock's team has analysed these smart contracts in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the 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.

Hashlock is not responsible for the safety of any funds and is not in any way liable for the security of the project.

Technical Disclaimer

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

About Hashlock

Hashlock is an Australian-based company aiming to help facilitate the successful widespread adoption of distributed ledger technology. Our key services all have a focus on security, as well as projects that focus on streamlined adoption in the business sector.

Hashlock is excited to continue to grow its partnerships with developers and other web3-oriented companies to collaborate on secure innovation, helping businesses and decentralised entities alike.

Website: hashlock.com.au
Contact: info@hashlock.com.au