hashlock.

Security Audit

Onyx DEX (dApp)

December 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

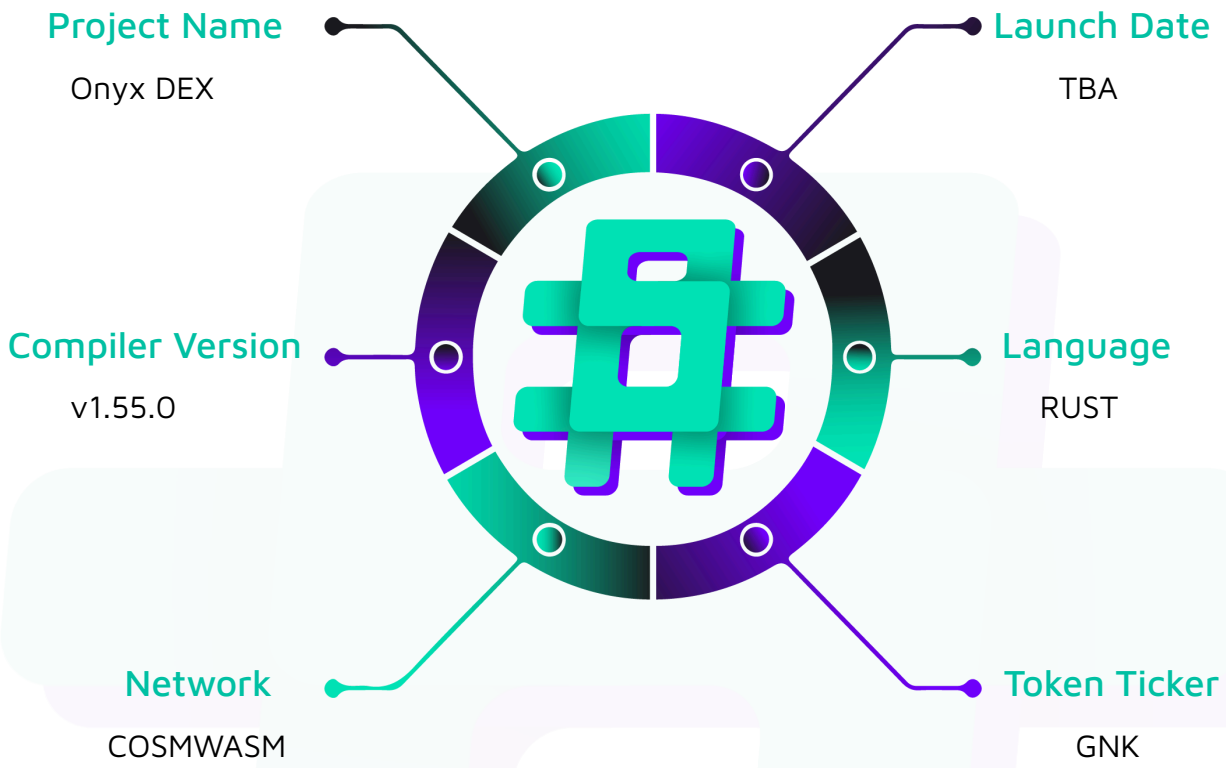
The Gonka DEX team partnered with Hashlock to conduct a security audit of their smart contracts. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts are secure.

Project Context

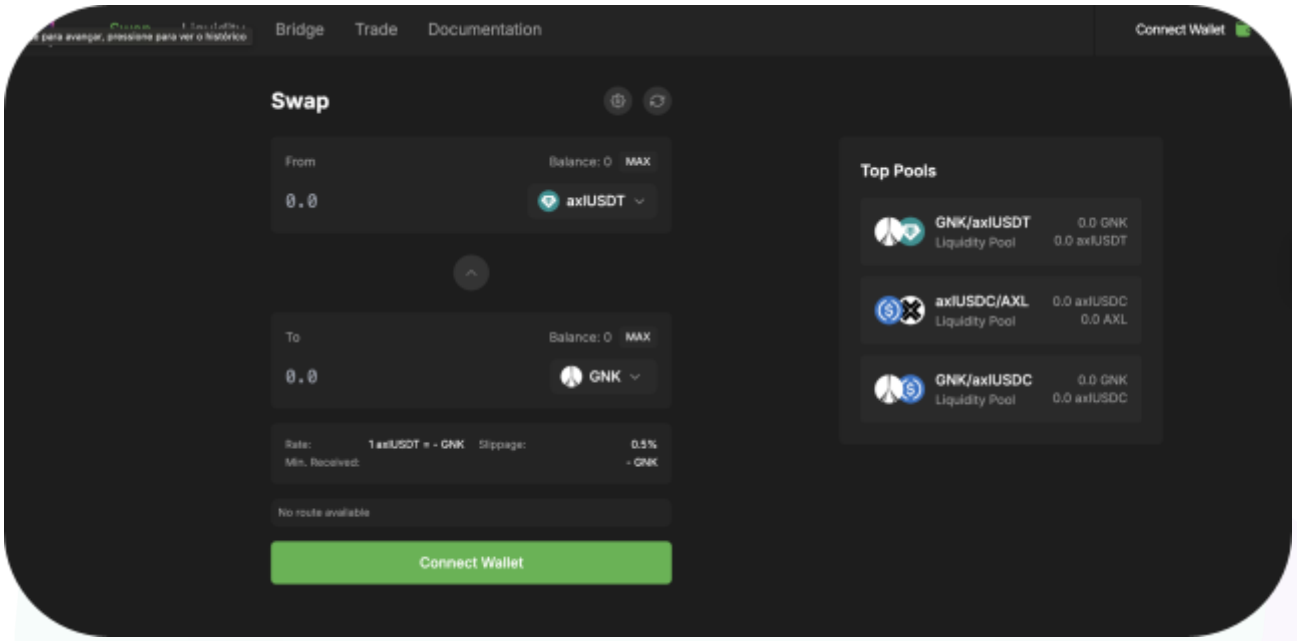
Onyx DEX is a decentralized exchange (DEX) that enables users to swap digital assets with low fees and high speed, powered by automated market-making and liquidity pools. The platform focuses on seamless token trading, providing liquidity incentives, transparent on-chain execution, and a user-friendly interface. By leveraging decentralized infrastructure, Onyx DEX removes intermediaries, allowing users to maintain full control of their assets while accessing efficient and permissionless swaps.

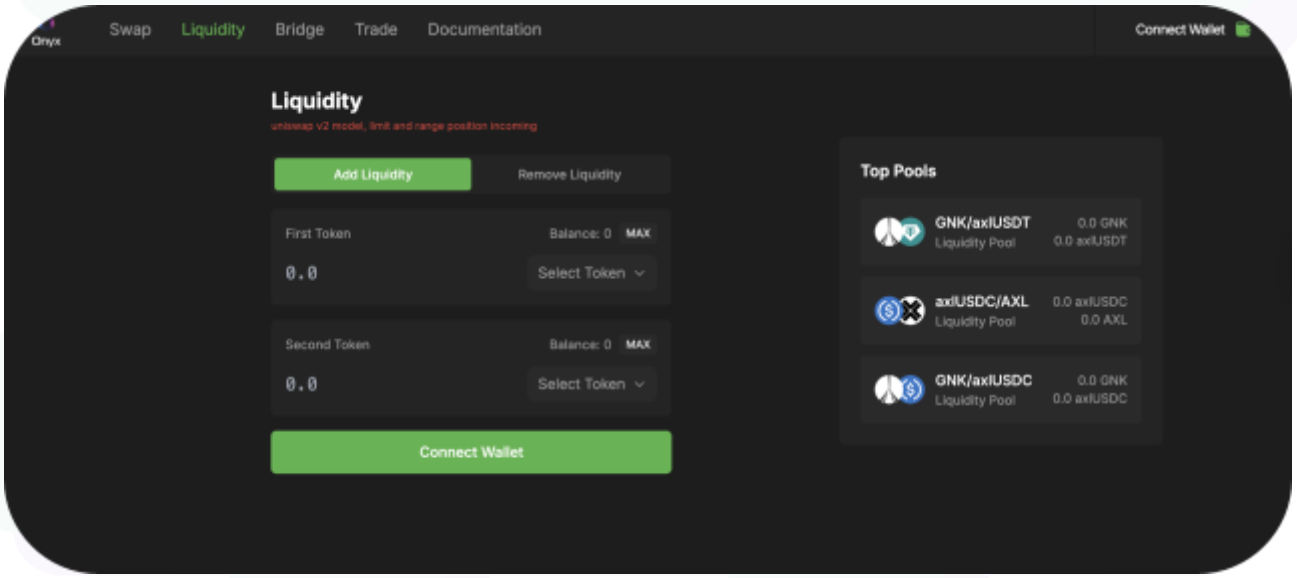
Project Name: Onyx DEX
Project Type: dApp
Compiler Version: 1.55.0
Website: https://www.getonyx.xyz
Logo:

Visualised Context:



Project Visuals:





Audit Scope

We at Hashlock audited the Rust code within the Onyx DEX 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	Onyx DEX Smart Contracts
Platform	CosmWasm / Rust
Audit Date	December, 2025
GitHub URL	https://github.com/Grand-Croix/gonka-dex/tree/main/contracts/contracts
In scope files	- terraswap_factory/ - src/contract.rs - src/lib.rs - src/state.rs - src/response.rs - terraswap_pair/ - src/contract.rs - src/lib.rs - src/state.rs - src/response.rs - terraswap_router/ - src/contract.rs - src/lib.rs - src/state.rs - src/operations.rs - terraswap_token/ - src/contract.rs - src/lib.rs
Audited GitHub Commit Hash	9fc2f8bee4ea44d1b5d76e1899721b8000fc8a68
Fix Review GitHub Commit Hash	ba7138610b3c3b7a69649f08347e2f2162e7cad7

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.

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 list of audited assets is presented in the Audit Scope section and the project's contract functionality is presented in the Intended Smart Contract Functions section.

All vulnerabilities initially identified have now been resolved.

Hashlock found:

1 High-severity vulnerability

3 Medium severity vulnerabilities

2 Low-severity vulnerabilities

1 QA

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

terraswap_factory
- Allows the owner to:
  - Update contract configurations
  - Update protocol fee configuration
  - Update native token decimals
  - Migrate pair contracts
- Allows users to:
  - Create liquidity pair contracts
	Contract achieves this functionality.

terraswap_pair
- Allows users to:
  - Provide liquidity
  - Withdraw liquidity
  - Swap funds
  - Collect protocol fees
	Contract achieves this functionality.

terraswap_router - Allows users to: - Swap native or CW20 tokens through a defined swap operation	**Contract achieves this functionality.**
terraswap_token - Source code for liquidity pool tokens created in pair contracts	**Contract achieves this functionality.**

Code Quality

This audit scope involves the smart contracts of the Onyx DEX 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 recommended to optimize security measures.

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 Onyx DEX 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

High

[H-01] terraswap_pair/src/contract.rs#provide_liquidity,withdraw_liquidity, swap - Protocol fees not deducted from pool calculations inflate pool liquidity

Description

Protocol fees are accumulated in the state via `PROTOCOL_FEES` during swaps, but are never deducted from pool balance queries. The `query_pools` function returns the total contract balance, including uncollected protocol fees.

This is problematic because the protocol fees are designated for the fee collector and should not contribute to the constant product `k`. However, they are incorrectly treated as part of the liquidity pool.

This issue exists in the `provide_liquidity`, `withdraw_liquidity`, and `swap` entry points.

Impact

Liquidity providers receive shares calculated against an inflated pool balance that includes protocol fees that should not be included. This causes LP providers to receive more funds than intended when redeeming liquidity, as it reflects accumulated protocol fees.

Additionally, swaps are executed against a larger liquidity pool than intended, resulting in an incorrect swap output amount. This may also prevent the protocol fees from being withdrawn due to insufficient funds.

Recommendation

Deduct accumulated protocol fees from pool balances in the `provide_liquidity`, `withdraw_liquidity`, and `swap` entry points.

Status

Resolved

Medium

[M-01]

terraswap_factory/src/contract.rs#execute_add_native_token_decimals - Native token balance verification allows theft of factory contract funds

Description

The `execute_add_native_token_decimals` function in `terraswap_factory/src/contract.rs:253` requires the factory contract to hold a non-zero balance of the native token being registered.

However, any funds held by the factory contract can be stolen by anyone who creates a pair with initial liquidity, as the `reply` handler transfers those funds to the newly created pair and mints LP tokens to the attacker in `terraswap_factory/src/contract.rs:335~359`.

Impact

Native tokens sent to the factory contract for decimal verification can be stolen by an attacker who:

1. Calls `execute_create_pair` with `Asset` structs specifying the amounts in the contract.
2. The `reply` handler transfers those funds from the factory contract to the new pair contract.
3. LP tokens are minted to the attacker's address.

Recommendation

Replace the balance verification mechanism with a native token supply query, or automatically refund the caller after successful verification.

Status

Resolved

[M-02] terraswap_factory/src/contract.rs#reply - Initial liquidity provision fails due to deadline parameter set to None

Description

When creating a pair with initial liquidity, the factory's `reply` handler calls `PairExecuteMsg::ProvideLiquidity` on the new pair contract with a `deadline` of `None`.

However, the pair contract's `assert_deadline` function disallows setting the `deadline` parameter to `None` in `terraswap_pair/src/contract.rs:991`, reverting with a `DeadlineRequired` error.

Impact

Creating pairs with initial liquidity is completely broken. Any call to `execute_create_pair` with non-zero asset amounts will fail during the `reply` phase, resulting in the entire transaction reverting.

Recommendation

Provide a reasonable `deadline` in the `PairExecuteMsg::ProvideLiquidity` call.

Status

Resolved

[M-03] terraswap_pair/src/contract.rs#swap - Zero return amount swap causes user fund loss

Description

When a swap results in a zero return amount (e.g., due to rounding, extreme pool imbalance, or tiny input amounts), the transaction succeeds silently rather than reverting, as seen in `contracts/contracts/terraswap_pair/src/contract.rs:573~575`.

This causes the user to lose their input tokens and receive nothing in return.

Impact

Users can unknowingly lose funds when swapping small amounts that round down after fees, or when swapping against extremely imbalanced pools. The swap transaction succeeds, protocol fees are accumulated, but the user receives nothing.

Recommendation

Revert the transaction when the return amount is zero.

Status

Resolved

Low

[L-01] terraswap_factory/src/contract.rs#execute_update_config - Single-step ownership transfer risks permanent lockout

Description

The `execute_update_config` function implements a single-step ownership transfer mechanism. If the owner accidentally transfers ownership to an invalid, inaccessible, or mistyped address, the factory contract becomes permanently locked with no way to recover administrative control.

Impact

If ownership is transferred to an address the owner does not control, the factory becomes permanently locked. Critical functions including `execute_update_config`, `execute_update_protocol_fee_config`, `execute_add_native_token_decimals`, and `execute_migrate_pair` become inaccessible.

Recommendation

Implement a two-step ownership transfer pattern that creates a pending ownership transfer, requiring the pending owner to accept it in a separate transaction.

Status

Resolved

[L-02] terraswap_pair/src/contract.rs#provide_liquidity - Incorrect refund_assets event emitted for CW20 tokens

Description

The `provide_liquidity` function emits a `refund_assets` event that includes refund amounts for both native tokens and CW20 tokens.

This is incorrect because only native tokens are actually refunded to the sender. CW20 tokens use `TransferFrom` to pull only the exact `desired_amount` (see `contracts/contracts/terraswap_pair/src/contract.rs:408`), so no refund occurs.

However, the emitted event incorrectly suggests CW20 tokens were refunded.

Impact

Off-chain systems, indexers, and UIs that rely on event data will display incorrect refund information for CW20 tokens, potentially causing user confusion.

Recommendation

Set CW20 refund amounts to zero in the emitted event.

Status

Resolved

QA

[Q-01] terraswap_router/src/contract.rs#assert_minimum_receive - Typo in parameter name minium_receive

Description

The `assert_minimum_receive` function contains a typo in its parameter name. The parameter is spelled `minium_receive` instead of the correct `minimum_receive`.

Recommendation

Rename the parameter to `minimum_receive`.

Status

Resolved

Centralisation

The Onyx DEX project values security and utility over decentralisation.

The owner executable functions within the protocol increase security and functionality but depend highly on internal team responsibility.

Conclusion

After Hashlock's analysis, the Onyx DEX project seems to have a sound and well-tested code base, now that our vulnerability findings have been resolved. 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