hashlock.

2024

Security Audit

Balanced (DeFi)

September 2024 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 THAT 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 THE USE OF THE CLIENT.

Executive Summary

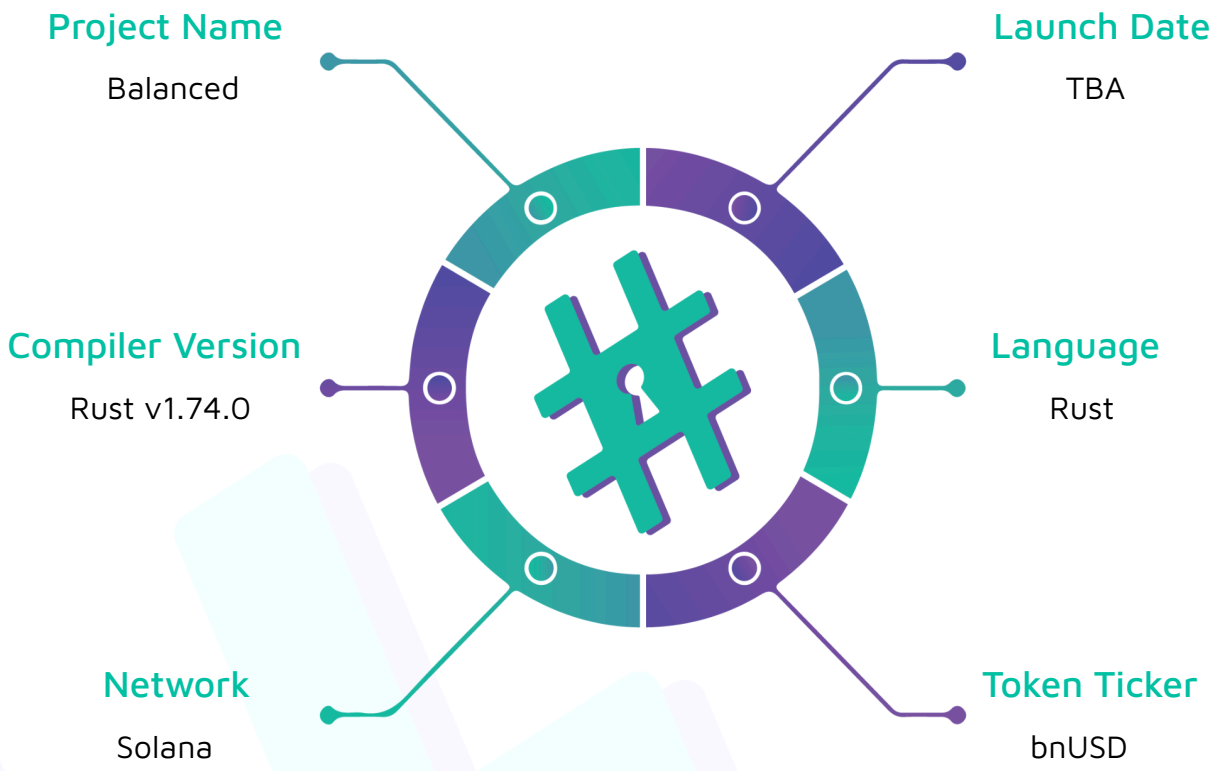
The Balanced team partnered with Hashlock to conduct a security audit of their Balanced protocol smart contracts. Hashlock manually and proactively reviewed the code to ensure the project's team and community that the deployed contracts were secure.

Project Context

The Balanced protocol is a decentralized platform that allows users to mint a stablecoin called Balanced Dollar (bnUSD) by depositing cryptocurrency as collateral. bnUSD is pegged to the US Dollar, and its value is maintained through over-collateralization, liquidation mechanisms, and the Stability Fund, which allows 1:1 swaps with other stablecoins.

Project Name: Balanced
Compiler Version: Rust v1.74.0
Website: https://balanced.network/
Logo:

Visualised Context:



Project Visuals:

Audit scope

We at Hashlock audited the Rust code within the Balanced 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	**Balanced Protocol Smart Contracts**
Platform	**Solana / Rust**
Audit Date	**August, 2024**
Repo 1	https://github.com/icon-project/xcall-multi
Components	• contracts/solana/programs/xcall/src/instructions/execute_call.rs • contracts/solana/programs/xcall/src/instructions/execute_rollback.rs • contracts/solana/programs/xcall/src/instructions/codec.rs • contracts/solana/programs/xcall/src/instructions/config.rs • contracts/solana/programs/xcall/src/instructions/fee.rs • contracts/solana/programs/xcall/src/instructions/handle_message.rs • contracts/solana/programs/xcall/src/instructions/send_message.rs • contracts/solana/programs/xcall/src/connection.rs • contracts/solana/programs/xcall/src/dapp.rs • contracts/solana/programs/xcall/src/helper.rs • contracts/solana/programs/xcall/src/lib.rs • contracts/solana/programs/xcall/src/state.rs • contracts/solana/programs/centralized-connection/src/instructions/query_accounts.rs • contracts/solana/programs/centralized-connection/src/contexts.rs • contracts/solana/programs/centralized-connection/src/helper.rs

	• contracts/solana/programs/centralized-connection/src/lib.rs • contracts/solana/programs/centralized-connection/src/state.rs
GitHub Commit Hash	49024516b766d129f09693fdf555f397bdd788e0
Repo 2	https://github.com/balancednetwork/Balanced-Solana-contracts
Components	• programs/xcall-manager/src/states.rs • programs/xcall-manager/src/lib.rs • programs/xcall-manager/src/instructions.rs • programs/xcall-manager/src/helpers.rs • programs/xcall-manager/src/configure_protocols.rs • programs/asset-manager/src/states.rs • programs/asset-manager/src/param_accounts.rs • programs/asset-manager/src/lib.rs • programs/asset-manager/src/instructions.rs • programs/balanced-dollar/src/states.rs • programs/balanced-dollar/src/param_accounts.rs • programs/balanced-dollar/src/lib.rs • programs/balanced-dollar/src/instructions.rs • programs/balanced-dollar/src/helpers.rs
GitHub Commit Hash	be0d8bd3cad39c9d864ac808d522d4eea82d1b51

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.

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.

We initially identified some significant vulnerabilities that have since been addressed.

Hashlock found:

2 High-severity vulnerabilities

3 Medium-severity vulnerabilities

3 Low-severity vulnerabilities

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 Behaviours

Claimed Behaviour	Actual Behaviour

XCall
- Allows users to:
  - Send cross-chain messages
  - Process incoming message requests and results
  - Process failed cross-chain messages and rollbacks
- Allows admins to:
  - Update protocol configurations
	Program achieves this functionality.

Centralized Connection
- Periphery contract used to:
  - Send, receive, and revert messages
- Allows admins to:
  - Update protocol configurations
  - Claim protocol fees
	Program achieves this functionality.

XCallManager
- Periphery contract used to:
  - Verify protocols used in the cross-chain bridging
  - Handle call messages
- Allows admins to:
  - Update protocol configurations
	Program achieves this functionality.

AssetManager
- Allows users to:
  - Send deposit messages with attached SPL or native tokens
  - Handle call messages
	Program achieves this functionality.

- Allows admins to: - Update protocol configurations	

Balanced Dollar
- Allows users to:
  - Transfer bnUSD tokens to other chains with message data
  - Handle call messages
	Program achieves this functionality.

Code Quality

This audit scope involves the smart contracts for the Balanced project, as outlined in the Audit Scope section. All contracts, libraries, and interfaces mostly follow standard best practices 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 Balanced projects 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 help understand the overall architecture of the protocol.

Dependencies

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

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

Audit Findings

High

[H-01] instruction.rs#deposit_token, deposit_native - Missing ownership validation allows users to fake deposits, causing a loss of funds for the protocol

Description

When a user transfers native or SPL tokens to other chains, no validation ensures the account deposited belongs to the protocol. This means that users could send the funds into their own accounts, tricking the protocol into thinking the deposit has been made.

Vulnerability Details

The `deposit_token` and `deposit_native` functions allow users to deposit SPL or native tokens and dispatch a message in `programs/asset-manager/src/instructions.rs:93-167`. However, no validation ensures that the funds are correctly deposited into accounts owned by the protocol.

In particular, the `ctx.accounts.vault_token_account` and `ctx.accounts.vault_native_account` accounts are not validated to be owned by the program. Users could specify another account they own and deposit into it, causing the protocol to receive zero funds even though the deposit transaction succeeded.

Impact

Users would receive a refund of tokens they never deposited, causing a loss of funds for the protocol. This occurs when the `handle_call_message` instruction is called in `programs/asset-manager/src/instructions.rs:289-293`.

Recommendation

Consider validating that the `ctx.accounts.vault_token_account` and `ctx.accounts.vault_native_account` accounts are owned by the program. For example, `ctx.accounts.vault_token_account` should be validated with `[b"vault",`

`mint.as_ref()]` seed constraint, and `ctx.accounts.vault_native_account` should be validated with `[b"vault_native"]` seed constraint.

Status

Resolved

[H-02] instruction.rs#handle_call_message - Missing PDA validation allows users to mint an arbitrary amount of tokens

Description

The `ctx.accounts.state` is not validated as the expected PDA owned by the program, allowing users to pass their malicious PDA with arbitrary parameters to bypass authentication.

Vulnerability Details

The `handle_call_message` function handles `CROSS_TRANSFER` and `CROSS_TRANSFER_REVERT` operations in `programs/balanced-dollar/src/instructions.rs:138-177`. It is intended only to be called by xCall from `programs/balanced-dollar/src/states.rs:64`, which then mints a specified amount of bnUSD token to the recipient.

However, the `state` account in `programs/balanced-dollar/src/states.rs:46` is not validated to be the intended PDA owned by the program. This allows users to impersonate xCall by passing a PDA with the `State.xcall` value as their controlled address, bypassing the signer validation.

Impact

Users can steal funds by minting any arbitrary amount of bnUSD token, triggering protocol insolvency as the minted tokens are not backed by sufficient funds.

Recommendation

Consider validating that `ctx.accounts.state` is the correct PDA by checking it with the `[b"state"]` seed constraint.

Status

Resolved

Medium

[M-01] instruction.rs#verify_protocols - Unchecked return values cause validation failure

Description

In several instances of the codebase, results from function validations are ignored. This causes the validation to be ineffective, potentially introducing security issues.

Vulnerability Details

The `verify_protocols` function in `programs/asset-manager/src/instructions.rs:240` calls `xcall_manager::cpi::verify_protocols` from the xCall manager program. However, the return boolean value that indicates whether the protocol is verified or not is ignored. This means that if the boolean returns as false, an error is not propagated to halt the transaction execution.

Note that validation results are also ignored in the following instances:

- `programs/balanced-dollar/src/instructions.rs:215`
- `programs/asset-manager/src/instructions.rs:493`
- `programs/asset-manager/src/instructions.rs:522`

Impact

The `handle_call_message` instruction can be executed even though the protocol validation has failed, which is incorrect as the transaction should revert to an error. Additionally, the `withdraw_token` and `withdraw_native_token` instructions can be executed even though the `AssetManagerError::ExceedsWithdrawLimit` error has been raised.

Recommendation

Consider returning an error if the validation has failed in the `handle_call_message`, `withdraw_token`, and `withdraw_native_token` instructions.

Status

Resolved

[M-02] Programs - Accounts are not validated to be the expected PDA

Description

In multiple instances of the codebase, user-supplied accounts are not enforced to be expected PDA owned by the program. This allows users to pass incorrect accounts with arbitrary parameters, potentially introducing security issues.

Vulnerability Details

PDA represents data accounts that hold state values and will be utilized by the program. It is important to validate them to ensure the stored values are correct.

The following accounts are not validated across the codebase:

- `pub token_state: Account<'info, TokenState>`
  - `programs/asset-manager/src/states.rs:50`
  - `programs/asset-manager/src/states.rs:115`
- `pub xcall_manager_state: Account<'info, xcall_manager::XmState>`
  - `programs/asset-manager/src/states.rs:68`
  - `programs/asset-manager/src/states.rs:128`
  - `programs/balanced-dollar/src/states.rs:29`
  - `programs/balanced-dollar/src/states.rs:58`
- `pub state: Account<'info, State>`
  - `programs/balanced-dollar/src/states.rs:74`
- `pub state: Account<'info, XmState>`
  - `programs/xcall-manager/src/states.rs:55`
  - `programs/xcall-manager/src/states.rs:64`
  - `programs/xcall-manager/src/states.rs:70`

Impact

Users may pass malicious PDA with arbitrary values to circumvent validations.

Recommendation

Consider validating the accounts to the expected PDA with the intended seed constraints.

Status

Resolved

[M-03] instruction.rs#handle_call_message - Insufficient validations when handling xCall messages

Description

The `handle_call_message` instruction does not perform sufficient validation to ensure the provided accounts match the expected address from the message data.

Vulnerability Details

The `handle_call_message` function in `programs/balanced-dollar/src/instructions.rs:116` does not perform sufficient validation to ensure the funds are sent to the correct recipient. Specifically, the recipient address (`ctx.accounts.to`) should be equal to `message.to` for the `CROSS_TRANSFER` operation, while for the `CROSS_TRANSFER_REVERT` operation, the recipient should be `message.account`.

Additionally, the `handle_token_call_message` function in `programs/asset-manager/src/instructions.rs:373-397` does not validate that the SPL token (`mint.key`) equals the intended token to be withdrawn (`message.token_address`).

Impact

Funds could be sent to incorrect recipient addresses, and SPL tokens could be sent with a different token than expected, causing a loss of funds scenario.

Recommendation

Consider implementing validations to ensure the accounts match the message's recipient and mint token addresses.

Status

Resolved

Low

[L-01] configure_protocols.rs - Duplicate constant defined

Description

The `CONFIGURE_PROTOCOLS` constant is defined in `programs/xcall-manager/src/configure_protocols.rs:10` and `programs/xcall-manager/src/helpers.rs:5` with the same variable name and value.

Recommendation

Consider only defining the constant once across the codebase to increase code readability and maintainability.

Status

Resolved

[L-02] Programs - Hardcoded seed inputs

Description

In multiple instances of the codebase, seed inputs are applied manually instead of referencing them with variables. For example, the `ctx.accounts.state` account validation depends on the hardcoded `"state"` seed constraint in `programs/xcall-manager/src/states.rs:7`, 33, and 40. This approach is problematic because an accidental typo would cause the supplied accounts to resolve into incorrect PDAs.

Recommendation

Consider defining constant variables for the seed values and reference them across the codebase instead of manually hardcoding them.

Status

Resolved

[L-03] helpers.rs#decode_cross_transfer_revert - Lack of validation of RLP data size

Description

The `decode_cross_transfer_revert` function in `programs/balanced-dollar/src/helpers.rs:49-66` does not validate the `rlp.item_count()` equals 3, which is expected as shown in `programs/balanced-dollar/src/structs/cross_transfer_revert.rs:15`.

Additionally, the `decode_deposit_revert_msg` function in `programs/asset-manager/src/helpers.rs:58-80` suffers from the same issue because it does not ensure `rlp.item_count()` equals 4, as shown in `programs/asset-manager/src/structs/deposit_revert.rs:16`.

Recommendation

Consider validating that the RLP data size matches the expected length in the `decode_cross_transfer_revert` and `decode_deposit_revert_msg` functions.

Status

Resolved

Centralisation

The Balanced 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 Hashlocks analysis, the Balanced 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 behavior 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 still need to verify the feasibility and impact of the issue. This process is vast because we document our suspicions early even if they are later shown not to 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 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


#Hashlock.