hashlock.

Security Audit

Glue (dApp)

July 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

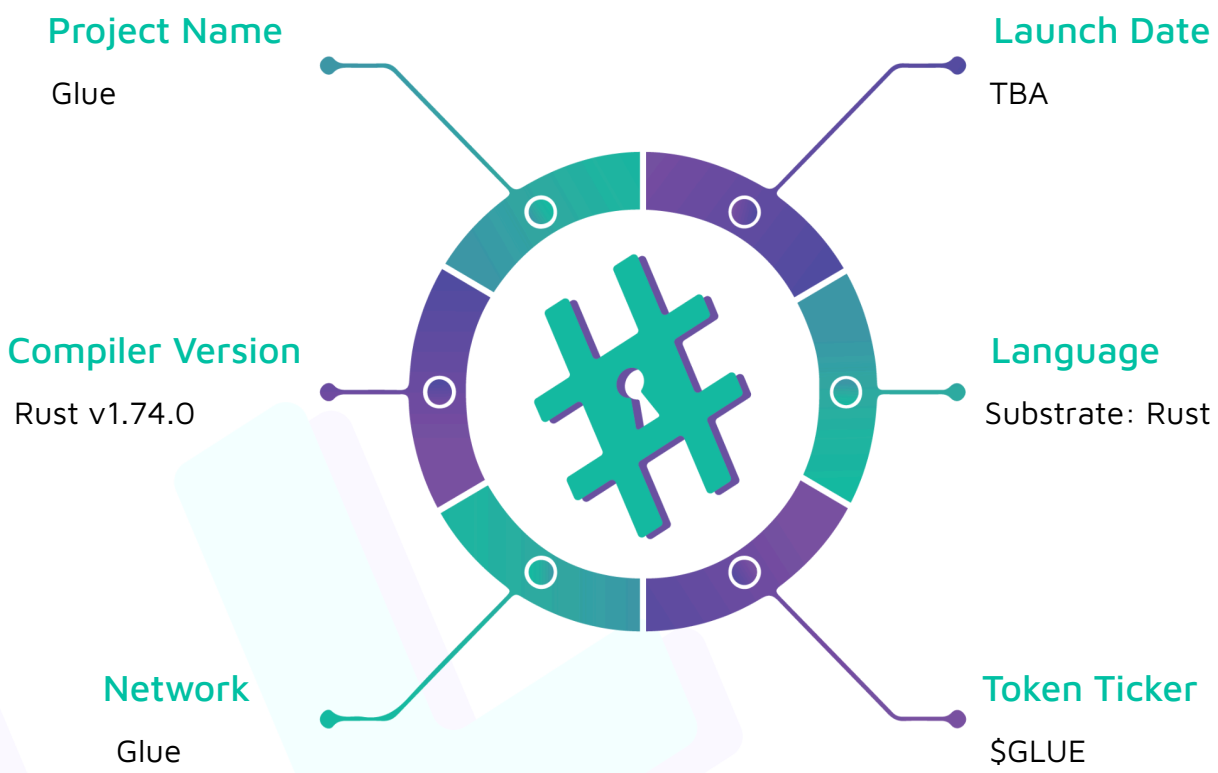
The Glue team partnered with Hashlock to conduct a security audit of their glue-l2-evm-smart-contracts-chain and glue-l1-relay-chain repositories. Hashlock manually and proactively reviewed the code to ensure the project's team and community that the deployed contracts were secure.

Project Context

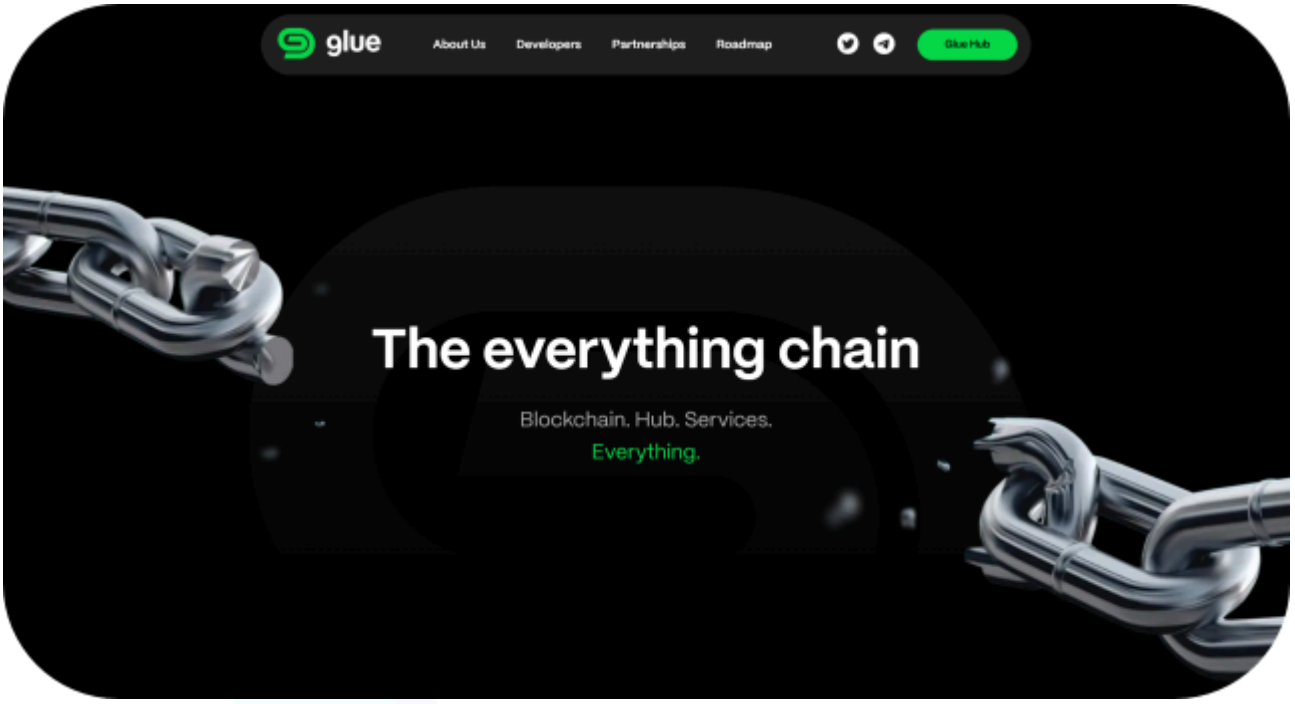
Glue is a blockchain ecosystem featuring a user-friendly interface and community-owned decentralized applications (dApps). It consists of a Substrate-based Layer 1 and three interconnected Layer 2s. The Glue token underpins the ecosystem, facilitating secure and affordable transactions. The project emphasizes fairness, avoiding highly discounted seed rounds

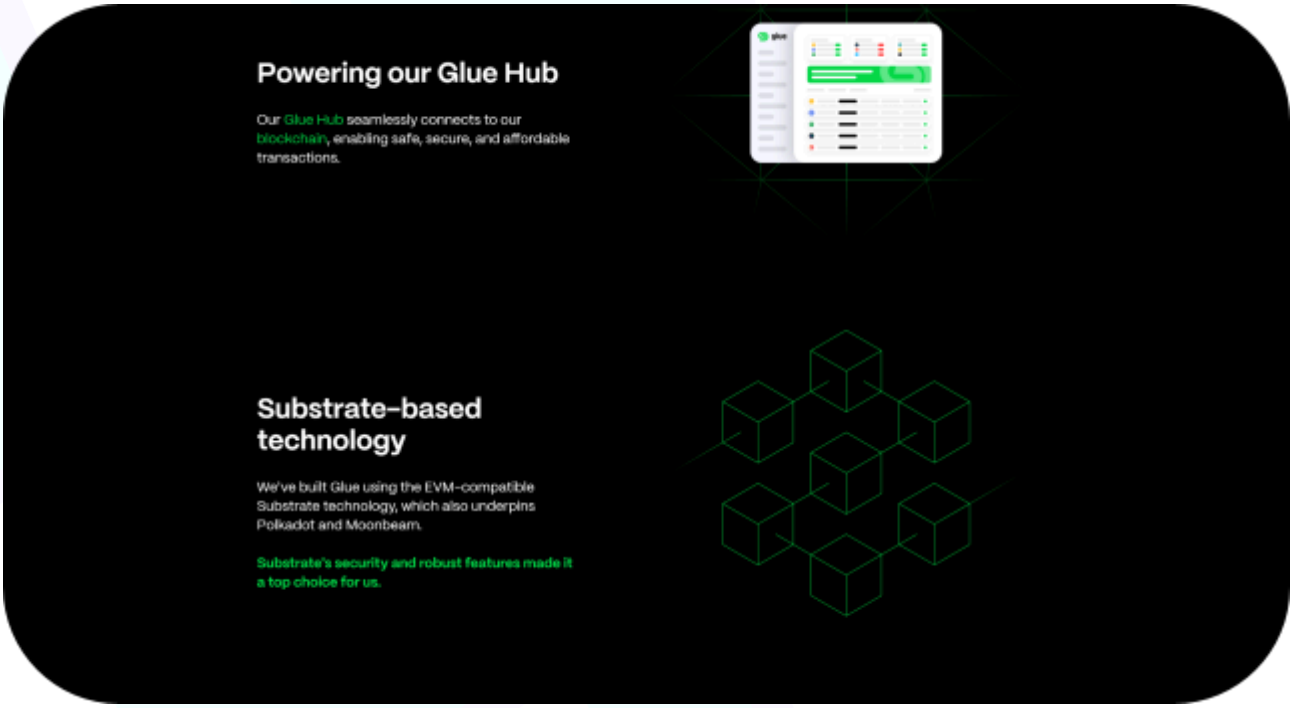
Project Name: Glue
Compiler Version: Rust v1.74.0
Website: https://glue.net
Logo:

Visualised Context:



Project Visuals:





Audit scope

We at Hashlock audited the Rust code within the Glue 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	**Glue Protocol Smart Contracts**
Platform	**Polkadot / Rust**
Audit Date	**July 8, 2024**
Repo 1	https://github.com/Glue-Net/glue-l2-evm-smart-contracts-chain
Component 1	/runtime/glue/*
Component 2	/precompiles/assets-erc20/*
Component 3	/pallets/subscriptions/*
Component 4	/node/service/src/chain_spec/glue.rs
Component 5	/node/service/src/chain_spec/mod.rs
Component 6	/node/service/src/client.rs
Component 7	/node/service/src/lib.rs
Repo 2	https://github.com/Glue-Net/glue-l1-relay-chain
Component 1	/cli/src/command.rs
Component 2	/node/service/src/chain_spec.rs
Component 3	/node/service/src/lib.rs

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.

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

Hashlock found:

1 High-severity vulnerabilities

0 Medium-severity vulnerabilities

5 Low-severity vulnerabilities

0 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 Behaviours

Claimed Behaviour	Actual Behaviour
glue-l2-evm-smart-contracts-chain **/runtime/glue** - Configurations for governance, XCM, ERC20 precompiles and others **/precompiles/assets-erc20** - Allows users to call ERC20 contract into pallets **/pallets/subscriptions** - Allows users to update subscriptions **/node/service/src/chain_spec/glue.rs** - Chain specifications for testnet and local network **/node/service/src/chain_spec/mod.rs** - Necessities for chain configurations **/node/service/src/client.rs** - Necessities for chain configurations **/node/service/src/lib.rs** - Necessities for chain configurations	**Pallet achieves this functionality.**
glue-l1-relay-chain **/cli/src/command.rs** - Necessities for command line interface **/node/service/src/chain_spec.rs** - Chain specifications for testnet and local network **/node/service/src/lib.rs** - Necessities for chain configurations	**Pallet achieves this functionality.**

Code Quality

This audit scope involves the smart contracts for the Glue 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 Glue 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 help us 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] assets-erc20#set_facilitator_bucket_capacity - Missing authentication in setFacilitatorBucketCapacity allows facilitators to bypass transfer restriction

Description

Facilitators can bypass the `bucket_capacity` restriction to transfer any amounts on behalf of the asset admin.

Vulnerability Details

The `set_facilitator_bucket_capacity` function in `precompiles/assets-erc20/src/lib.rs:698` does not validate that the caller must be the asset ID admin when configuring the facilitator's bucket capacity. This allows facilitators to bypass the `FACILITATOR_BUCKET_CAPACITY_EXCEEDED` validation in `precompiles/assets-erc20/src/lib.rs:331-335` by setting the bucket capacity to zero value, thereby enabling them to transfer any amount of funds on behalf of the asset admin.

Impact

Facilitators can transfer an arbitrary amount of tokens on behalf of asset admin without being restricted by `bucket_capacity`.

Recommendation

Consider checking the caller to be asset admin in the `set_facilitator_bucket_capacity` function.

Status

Resolved

Low

[L-01] pallet-parachain-onboarding#onboard_parachain - Duplicate parachain ID can be configured

Description

The `onboard_parachain` function in `pallets/pallet-parachain-onboarding/src/lib.rs:99-104` appends a new parachain (`para_id`) to the list of parachains (`<Parachains<T>>`) and updates all parachains. However, it does not check whether the parachain is already inside the list of parachains.

Recommendation

As no business logic requirement is needed for duplicate parachains, consider only adding the parachain if it is not one of the existing parachains.

Status

Resolved

[L-02] Pallets - Outstanding TODO comments

Description

The codebase features several TODO comments that indicate the codebase is under active development and not ready for production.

Recommendation

Consider resolving the TODO comments or removing them.

Status

Acknowledged

[L-03] subscriptions#update_subscription - Extinct should emit `updates` also

Description

It was found that `update_subscription` extinct emits an event after successful execution, returning origin, which signed the transaction. While the main functionality is adding new updates to subscriptions, it could be valuable to also return information about what has been removed (`Add`) or added (`Remove`).

Recommendation

We recommend that the `updates` parameter be included in the list of attributes returned by the emitter.

Status

Resolved

[L-04] runtime - Not used traits imported within the runtime

Description

It was found that in `runtime/glue/src/lib.rs` multiple traits are imported, and never used. The likely cause is that Moonbeam code was copied and adapted for Glue, and unnecessary imports were left.

This affects the readability of the code and its size.

Recommendation

We recommend adjusting imports within the entire codebase so that they only contain traits, types, and functions used in a given file.

Status

Resolved

[L-05] runtime - Typo in PalletXcmExtrinsiscsBenchmark

Description

It was found that in `runtime/glue/src/lib.rs:1589`, `PalletXcmExtrinsiscsBenchmark` is misspelled because it contains a typo. In the default runtime version, this benchmark is called `PalletXcmExtrinsicsBenchmark`.

Recommendation

We suggest adjusting the benchmark name so that it is linguistically correct.

Status

Resolved

Centralisation

The Glue 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 Glue 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.