hashlock.

Security Audit

Diffuse Protocol (DeFi)

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 Diffuse Protocol 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

Diffuse Prime is a non-custodial protocol that enables hedge funds to amplify yields by up to 3x through overcollateralized loans funded by Diffuse Prime LPs.

LP capital is deployed into hedge fund DeFi strategies with lending-market-grade security, enhanced by verifiable risk assessment and an automated capital protection layer.

Diffuse Protocol powers Diffuse Prime with a trustless, multichain infrastructure built on TEE, zkTLS Oracles, Collateral Abstraction, and zkMessaging, providing cryptographically verifiable trust and security.

Project Name: Diffuse Protocol
Project Type: Defi
Compiler Version: 0.8.30
Website: https://test.prime.diffuse.fi/
Logo:

Diffuse Prime

Visualised Context:

Project Name

Diffuse Protocol

Launch Date

TBA

Compiler Version

0.8.30

Language

SOLIDITY

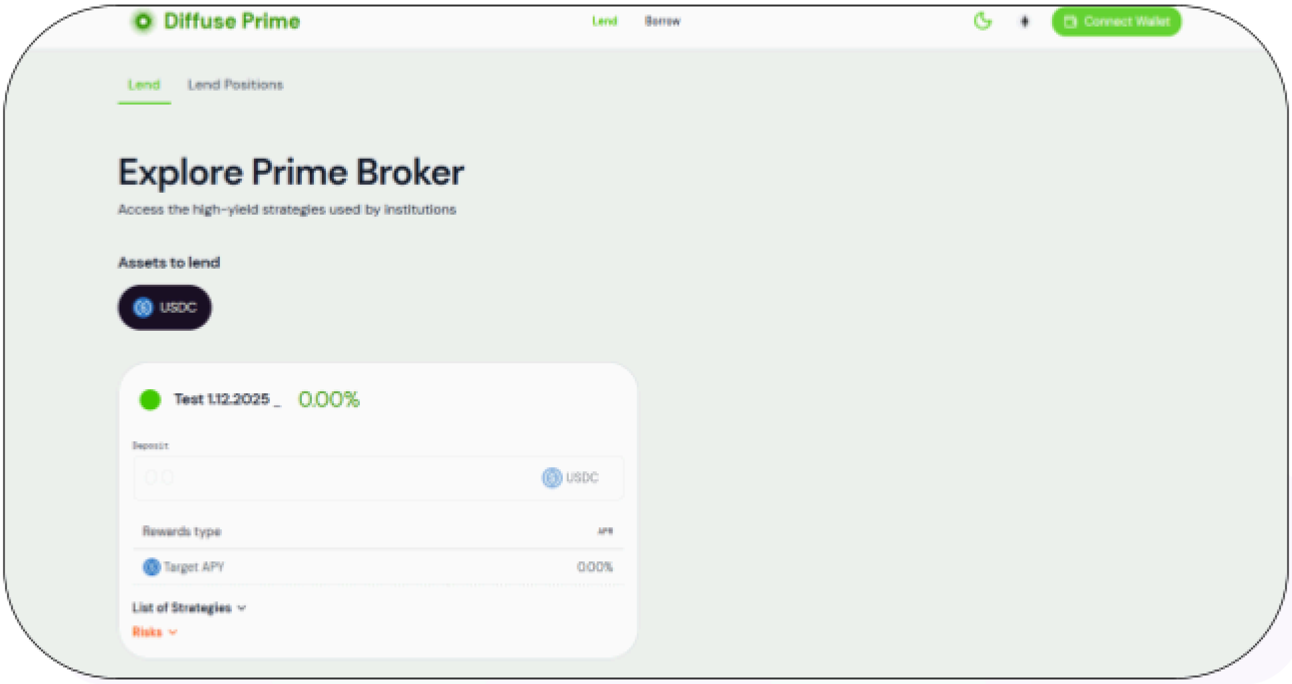
Network

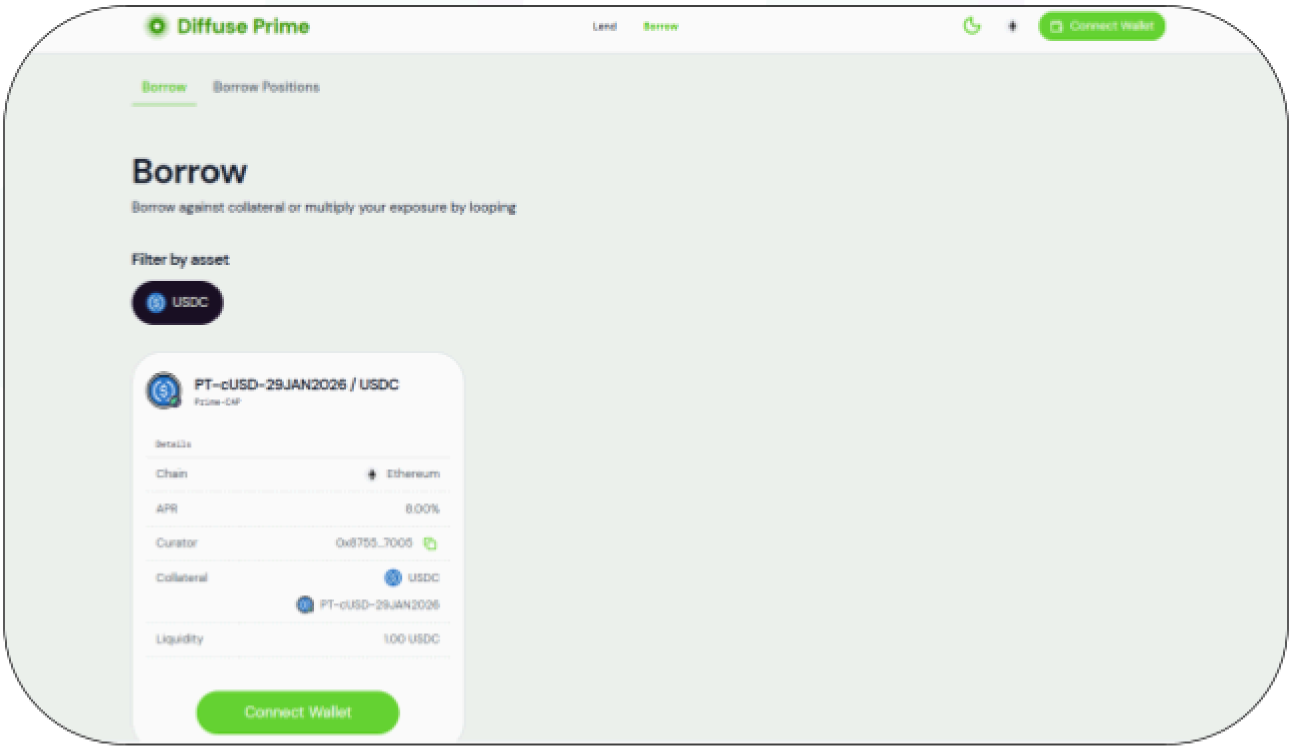
ETHEREUM

Token Ticker

TBA

Project Visuals:





Audit Scope

We at Hashlock audited the solidity code within the Diffuse Protocol 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	Diffuse Protocol Smart Contracts
Platform	Ethereum / Solidity
Audit Date	December, 2025
Contract 1	CapBurnAdapter.sol
Contract 2	CapMintAdapter.sol
Contract 3	DepositErc4626Adapter.sol
Contract 4	RedeemErc4626Adapter.sol
Contract 5	MintAdapter.sol
Contract 6	RedeemInfinifyAdapter.sol
Contract 7	AdapterWhiteList.sol
Audited GitHub Commit Hash	6619e03cbb5032803144d1c803497135eeaab459
Fix Review GitHub Commit Hash	ddcd3f50ba065c2acff0bd0be3b1bc4b69c5478a

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.

Not Secure	Vulnerable	Secure	Hashlocked

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 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
CapBurnAdapter.sol - Burns TOKEN_IN via external burner - Returns TOKEN_OUT to caller - Uses minOut protection	**Contract achieves this functionality.**
CapMintAdapter.sol - Mints TOKEN_OUT using TOKEN_IN via external minter - Enforces whitelist on callers - Uses minOut protection	**Contract achieves this functionality.**
DepositErc4626Adapter.sol - Deposits TOKEN_IN into ERC4626 vault - Mints vault shares (TOKEN_OUT) to caller - Uses minOut protection	**Contract achieves this functionality.**
RedeemErc4626Adapter.sol - Redeems ERC4626 shares (TOKEN_IN) for underlying (TOKEN_OUT) - Returns underlying to caller - Uses minOut protection	**Contract achieves this functionality.**
MintAdapter.sol (Infinify) - Mints TOKEN_OUT via external minter using TOKEN_IN - Uses minOut protection	**Contract achieves this functionality.**
RedeemInfinifyAdapter.sol - Redeems TOKEN_IN via Infinify gateway for TOKEN_OUT - Uses user-provided minOut	**Contract achieves this functionality.**

AdapterWithWhitelist.sol - Provides vault whitelist mechanism for adapters - Owner can add/remove vaults - Exposes read accessors	**Contract achieves this functionality.**

Code Quality

This audit scope involves the smart contracts of the Diffuse Protocol 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 were 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 Diffuse Protocol 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

QA

[Q-01] AdapterWithWhitelist#setVaultWhitelisted- misleading event emission on no-op state change

Vulnerability Details

The `setVaultWhitelisted` function emits a `VaultRemovedFromWhitelist` event even when the vault address was never previously whitelisted. This occurs when the function is called with `whitelisted = false` for an address that is already unset, causing an event that does not reflect an actual state transition.

```
function setVaultWhitelisted(address vault, bool whitelisted) external onlyOwner {
    isVaultWhitelisted[vault] = whitelisted;
    if (whitelisted) {
        emit VaultWhitelisted(vault, whitelisted);
    } else {
        emit VaultRemovedFromWhitelist(vault);
    }
}
```

Impact

Off-chain consumers may interpret a removal event as a real whitelist change, causing operational confusion, incorrect `UI` state, or inaccurate monitoring assumptions even though the on-chain whitelist did not change.

Recommendation

We recommend returning early when no state change occurs before emitting events.

Status

Resolved

Centralisation

The Diffuse Protocol 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 Diffuse Protocol 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