d3ploy

d3ploy

SECURITY ASSESSMENT

Radiate Protocol

October 27th 2023

TABLE OF

Contents

LEGAL

Disclaimer

D3ploy audits are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts d3ploy to perform a security review. D3ploy does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

D3ploy audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort. The report is provided only for the contract(s) mentioned in the report and does not include any other potential additions and/or contracts deployed by Owner. The report does not provide a review for contract(s), applications and/or operations, that are out of this report scope.

D3ploy's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

D3ploy represents an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. D3ploy's position is that each company and individual are responsible for their own due diligence and continuous security. The security audit is not meant to replace functional testing done before a software release. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent manual audits and a public bug bounty program to ensure the security of the smart contracts.

D3PLOY

Introduction

D3ploy is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

Secure your project with d3ploy

We offer field-proven audits with in-depth reporting and a range of suggestions to improve and avoid contract vulnerabilities. Industry-leading comprehensive and transparent smart contract auditing on all public and private blockchains.

Vunerability checking

A crucial manual inspection carried out to eliminate any code flaws and security loopholes. This is vital to avoid vulnerabilities and exposures incurring costly errors at a later stage.

Contract verification

A thorough and comprehensive review in order to verify the safety of a smart contract and ensure it is ready for launch and built to protect the end-user

Risk assessment

Analyse the architecture of the blockchain system to evaluate, assess and eliminate probable security breaches. This includes a full assessment of risk and a list of expert suggestions.

In-depth reporting

A truly custom exhaustive report that is transparent and depicts details of any identified threats and vulnerabilities and classifies those by severity.

Fast turnaround

We know that your time is valuable and therefore provide you with the fastest turnaround times in the industry to ensure that both your project and community are at ease.

Best-of-class blockchain engineers

Our engineers combine both experience and knowledge stemming from a large pool of developers at our disposal. We work with some of the brightest minds that have audited countless smart contracts over the last 4 years.

PROJECT

Introduction

Radiate Protocol is a yield enhancing protocol built on top of Radiant Capital. Radiate allows Radiant depositors to earn $RDNT emissions without locking any of their own dLP. On the other hand, Radiant allows liquidity providers earn boosted yields on dLP with no lockups.

Stakers can do single sided staking for DLP and earn WETH denominated yields. No need to worry about claiming multiple tokens or worrying about locking for certain durations.

Project Name	*Radiate Protocol*
Contract Name	*RADT Token*
Contract Address	*0x7ca0b5ca80291b1feb2d45702ffe56a7a53e7a97*
Contract Chain	*Mainnet*
Contract Type	*Smart Contract*
Platform	*EVM*
Language	*Solidity*
Network	*Arbitrium*
Codebase	*Private GitHub Repository*
Total Supply	*200,000*

INFO

Social

https://www.radiateprotocol.com/

https://twitter.com/RadiateProtocol

-

https://discord.gg/Ahg3hDGPCQ

https://docs.radiateprotocol.com/

https://github.com/RadiateProtocol

-

79

PASS

AUDIT

Score

Issues	8
Critical	2
Major	0
Medium	1
Minor	1
Informational	2
Discussion	2

All issues are described in further detail on the following pages.

AUDIT Scope

GITHUB REPOSITORY	LOCATION
RadiateProtocol/dlp-vaults/blob/update/audit_09_01/src/policies/Leverager_Audit.sol	✦ GitHub Repository

REVIEW Methodology

TECHNIQUES

This report has been prepared for Radiate Protocol to discover issues and vulnerabilities in the source code of the Radiate Protocol project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic, Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective in the comments below.

TIMESTAMP

Version	v1.0
Date	2023/10/27
Descrption	Layout project
	Architecture / Manual review / Static & dynamic security testing
	Summary

KEY Finding

TITLE	SEVERITY	STATUS
Potential Protocol Fund Drain	✦ Criticial	Pending
Funds Drained Due to Bypass of Unstakeable Amount Validation	✦ Criticial	Pending
Unrecoverable Locked Rewards in claimVested()	✦ Medium	Pending
Outdated Pragma Version	✦ Minor	Pending
Missing NatSpec Comments	✦ Informational	Pending
Require with Empty Message	✦ Informational	Pending
Public Constants can be Private	✦ Gas	Pending
Gas Optimization for State Variables	✦ Gas	Pending

IN - DEPTH Vulnerabilities

DESCRIPTION

The smart contract exhibits a vulnerability related to funds draining from the protocol. This issue arises from the interaction between the claim() and claimVested() functions. When a user first calls claim(), it sets info.pending to zero. However, when the same user later calls claimVested(), they can effectively withdraw the previously claimed amount again. Afterward, the user can call claim() again, which internally calls _update(), resetting info.pending to the same value as during the first withdrawal. This situation allows a user to drain funds from the protocol.

AFFECTED CODE

- *Leverager.claim()* - Leverager_Audit.sol L708-L731
- *Leverager.claimVested()* - Leverager_Audit.sol L749-L791
- *Leverager._update()* - Leverager_Audit.sol L416-L456

Issue : Potential Protocol Fund Drain

Level : Critical

Type : Fund Drain

Remediation : To mitigate this vulnerability, it is recommended to add stricter checks within the claim() and claimVested() functions to prevent multiple claims of the same rewards by the same user. A user should not be able to make multiple claims of the same reward, thus safeguarding the protocol's funds. Additionally, careful review and testing are essential to ensure the correctness of the proposed changes.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

The contract includes an unstake() function that allows users to withdraw their staked tokens, with a validation check to ensure that the amount to be withdrawn does not exceed the "unstakeable" amount. However, there is a vulnerability in the contract that allows an attacker to bypass this validation and drain funds from the contract.

The vulnerability arises from the _unloop() function, which is called within unstake(). In _unloop(), a flash loan is initiated through Aave's AAVE_LENDING_POOL to repay the debt, withdraw funds, and perform other operations. When the Aave flash loan is executed, it calls the executeOperation() function. In this callback, the contract repays the debt and withdraws the amount to be paid back to Aave.

An attacker can exploit this flow by directly calling the Aave flash loan function and providing the address of the contract (dlpVault) to be called back. The attacker can also supply a loan amount greater than the "unstakeable" amount. This way, they can bypass the unstakeable amount validation within the unstake() function. As a result, the attacker can drain more funds from the contract than unstakeableamoungt , causing a potential financial loss to the protocol.

AFFECTED CODE

- *Leverager.unstake()* - Leverager_Audit.sol L646-L661
- *Leverager.unloop()* - Leverager_Audit.sol L547-L588
- *dlpVault.executeOperation()* - DLPVault_Audit.sol L517-L575

Issue : Funds Drained Due to Bypass of Unstakeable Amount Validation

Level : Critical

Type : Fund Drain

Remediation : To mitigate this vulnerability, it's essential to ensure that the unstakeable amount validation is enforced for all fund withdrawals. Additionally, it is recommended to review and modify the flash loan mechanism to prevent unauthorized withdrawals and protect the contract from potential exploits.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

The contract allows users to make claims for their rewards using the claim() function. This function sets the expireAt timestamp for the claim, info.pending to zero and after this time, the claim can no longer be processed. However, the claimVested() function has a condition that checks if info.expireAt is greater than or equal to the current block's timestamp. If this condition is not meet, the claimVested() function does not allow the withdrawal of rewards, which is reasonable in preventing claims for which the expiration period has passed.

The issue arises when the expireAt timestamp has been set and the user misses the claim or decides not to claim the rewards immediately. In such cases, if the expireAt timestamp has passed, there is no way to recover or restart the claim. The user's rewards will be effectively stuck in the contract forever.

AFFECTED CODE

- *Leverager.claim()* - Leverager_Audit.sol L708-L731
- *Leverager.claimVested()* - Leverager_Audit.sol L749-L791

Issue : Unrecoverable Locked Rewards in claimVested()

Level : Medium

Type : Locked Funds

Remediation : To address this issue, consider implementing a mechanism that allows users to restart the claim process or recover their rewards if the expireAt timestamp has passed. This can be achieved through a time-lock mechanism that allows users to unlock their rewards after the initial expiration, with a penalty or time delay.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

Using an outdated compiler version can be problematic, especially if there are publicly disclosed bugs and issues that affect the current compiler version.
The contracts found in the repository were allowing an old compiler version to be used, i.e., 0.8.15.

AFFECTED CODE

- Leverager_Audit.sol L02

Issue : Outdated Pragma Version

Level : Minor

Type : Outdated Pragma

Remediation : Keep the compiler versions updated in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.20 or 0.8.21 pragma version which is stable and not too recent.

Reference: *https://swcregistry.io/docs/SWC-103*

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

Solidity contracts use a special form of comments to document code. This special form is named the Ethereum Natural Language Specification Format (NatSpec). The document is divided into descriptions for developers and end-users along with the title and the author.
The contracts in the scope were missing these comments

AFFECTED CODE

- Leverager_Audit.sol L02

Issue : Missing NatSpec Comments

Level : Informational

Type : Missing Best Practices

Remediation : Developers should review their codebase and add Natspec comments to all relevant functions, variables, and events. Natspec comments should include a description of the function or event, its parameters, and its return values.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

During analysis; multiple require statements were detected with empty messages. The statement takes two parameters, and the message part is optional. This is shown to the user when and if the require statement evaluates to false. This message gives more information about the conditional and why it gave a false response.

AFFECTED CODE

- Leverager_Audit.sol L528

Issue : Require with Empty Message

Level : Informational

Type : Code Optimization

Remediation : It is recommended to add a descriptive message, no longer than 32 bytes, inside the require statement to give more detail to the user about why the condition failed.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.

AFFECTED CODE

- Leverager_Audit.sol L40-L63

Issue : Public Constants can be Private

Level : Gas

Type : Gas Optimization

Remediation : If reading the values for the constants is not necessary, consider changing the public visibility to private.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

In Solidity, the compound assignment operators '+=' and '-=' tend to consume more gas compared to the basic addition and subtraction operators ('+' and '-', respectively). As a result, when you use 'x += y', it typically incurs a higher gas cost than using 'x = x + y'.

AFFECTED CODE

- Leverager_Audit.sol L513-L514, L583-L587, L836-L840

Issue : Gas Optimization for State Variables

Level : Gas

Type : Gas Optimization

Remediation : Replace += and -= with the basic + and - operators whenever feasible. This can help reduce gas consumption, especially when working with large-scale operations.

Alleviation / Retest :

SOURCE Code

GitHub Repository

RadiateProtocol/dlp-vaults/blob/update/audit_09_01/src/policies/Leverager_Audit.sol

REPORT Appendix

FINDING CATEGORIES

The assessment process will utilize a mixture of static analysis, dynamic analysis, in-depth manual review and/or other security techniques.

This report has been prepared for Radiate Protocol project using the above techniques to examine and discover vulnerabilities and safe coding practices in Radiate Protocol's smart contract including the libraries used by the contract that are not officially recognized.

A comprehensive static and dynamic analysis has been performed on the solidity code in order to find vulnerabilities ranging from minor gas optimizations to major vulnerabilities leading to the loss of funds.

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The testing methods find and flag issues related to gas optimizations that help in reducing the overall gas cost It scans and evaluates the codebase against industry best practices and standards to ensure compliance It makes sure that the officially recognized libraries used in the code are secure and up to date.

AUDIT SCORES

D3ploy Audit Score is not a live dynamic score. It is a fixed value determined at the time of the report issuance date.

D3ploy Audit Score is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports and scores are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts d3ploy to perform a security review.

d3ploy

WEBSITE d3ploy.co

@d3ploy_ TWITTER