PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Derive

20 November 2024

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Derive contracts on the Derive network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Derive
URL	https://github.com/derivexyz/derive-governance
Platform	Derive
Language	Solidity
Preliminary Contracts	https://github.com/derivexyz/derive-governance/blob/0614c6526cf1299946a227732d9977523cc1e4f3/src/token/StakedDeriveToken.sol
Resolution	https://github.com/derivexyz/derive-governance/blob/27e32a253551a2ab6123e8558565085c7706188a/src/StakedDeriveToken.sol

1.2 Contracts Assessed

Name	Contract	Live Code Match
`StakedDeriveToken`	`Proxy: 0x7499d654422023a407d92e1D83D387d81BC68De1` `Implementation: 0xc3d960B2D0A1d23b1d2073293fc80625d7Fa1fbc`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
Governance	0	-	-	-
High	4	3	-	1
Medium	1	-	-	1
Low	4	2	-	2
Informational	3	-	2	1
Total	**12**	**5**	2	**5**

Classification of Issues

Severity	Description
Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 StakedDeriveToken

ID	Severity	Summary	Status
01	HIGH	`_authorizeUpgrade` is not permissioned	✓ RESOLVED
02	HIGH	Inefficient whitelist transfer restrictions	ACKNOWLEDGED
03	HIGH	Burning of `deriveToken` will fail, resulting in loss of funds	✓ RESOLVED
04	HIGH	Users can mint as many voting units as they want to themselves	✓ RESOLVED
05	MEDIUM	Precision loss due to ratio being in 0 decimals	ACKNOWLEDGED
06	LOW	`updateDividendsAddress` could result in loss of accrued dividends	ACKNOWLEDGED
07	LOW	Approved usage ambiguous functionality	ACKNOWLEDGED
08	LOW	`redeem` can have a duration greater than `maxRedeemDuration`	✓ RESOLVED
09	LOW	The default values for several variables will not be applicable	✓ RESOLVED
10	INFO	Implementation getter not available	ACKNOWLEDGED
11	INFO	Typographical issues	PARTIAL
12	INFO	Gas optimizations	PARTIAL

2 Findings

2.1 StakedDeriveToken

`StakedDeriveToken` is an escrowed governance token of the DRV token. It allows users to stake their DRV tokens, earn governance rights, and allocate their staked tokens to various usages for additional benefits.

The flexible redemption mechanism allows users to customize their vesting periods, balancing liquidity needs and rewards. Integrated dividends support ensures users continue to earn during vesting, improving the incentives for the users.

Conversion to `stDRV` from `DRV`:

- Users can stake their `DRV` tokens by converting them into `stDRV` at a `1:1` ratio using the `convert` function.
- The `convertTo` function allows conversion to a specific address, enabling staking on behalf of others.

Vesting Period:

- Redemption can be immediate or over a period ranging from `minRedeemDuration` to `maxRedeemDuration`.
- Longer vesting durations yield higher `DRV` return ratios.

Usage Approval:

- Users can approve specific usages (external contracts) to allocate a certain amount of their `stDRV` via `approveUsage`.

Allocation and Deallocation:

- `allocate` function locks `stDRV` tokens in the contract for use by the approved usage.
- `deallocate` function allows users to withdraw their `stDRV` from a usage.

- Allocations enable users to participate in various activities or earn rewards through plugins.

2.1.1 Privileged Functions

- `updateRedeemSettings`
- `updateDividendsAddress`
- `updateDeallocationFee`
- `updateTransferWhitelist`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	`_authorizeUpgrade` is not permissioned
Severity	HIGH SEVERITY
Description	The contract uses OpenZeppelin's UUPS proxy. This proxy has a function called `_authorizeUpgrade` that is used to upgrade the proxy with new implementations. This function is not permissioned by default and OpenZeppelin explicitly states that the derived contracts should override and add a permission to this function. As `_authorizeUpgrade` is not permissioned, anyone will be able to upgrade the UUPS proxy.
Recommendation	Consider adding an `onlyOwner` modifier to the function.
Resolution	RESOLVED The protocol now uses the transparent proxy pattern.

Issue #02	Inefficient whitelist transfer restrictions
Severity	HIGH SEVERITY

Description	Within `_update()`, there is the following check:
	`require(from == address(0) ||`
	`_transferWhitelist.contains(from) ||`
	`_transferWhitelist.contains(to), "transfer: not allowed");`
	
	Non-whitelisted users can transfer to whitelisted users, whitelisted users can transfer to non-whitelisted users and non-whitelisted users can transfer `stDrv` to non-whitelisted users.

Recommendation	Always revert if the `to` address is not whitelisted and handle the cases where `from`/`to` is `address(0)`.
Resolution	ACKNOWLEDGED The team has stated that the non-whitelisted to whitelisted and whitelisted to non-whitelisted transfers are intended. We must mention that a non-whitelisted can transfer to another non-whitelisted if they use the `convertTo` function. This will be possible only if the non-whitelisted sender converts DRV to stDRV directly to another non-whitelisted address.

Issue #03	**Burning of `deriveToken` will fail, resulting in loss of funds**
Severity	HIGH SEVERITY
Location	`// burns DRV excess if any` `deriveToken.burn(address(this),deriveExcess);`
Description	Throughout the contract, `deriveToken` is burnt, for example in `_finalizeRedeem`. The issue resides in the fact that this function does not exist in `deriveToken` which will result in a revert. Unfortunately this does not always happen—`_finalizeRedeem` will revert only if an excess of `deriveToken` exists which will revert and stop the user from finalizing the redeem.
Recommendation	Consider implementing the correct `burn` function.
Resolution	RESOLVED The contract now transfers the amount that was previously burned to a `feeRecipient` controlled by the project.

Issue #04	Users can mint as many voting units as they want to themselves
Severity	HIGH SEVERITY

Description	Within `_update()`, `_transferVotingUnits()` is skipped if the receiver of the tokens is the contract itself. This was done with the intention of the contract to not hold voting units when users are allocating.
	
	However, this allows users to mint as many voting units to themselves since they can call `redeem()` and `cancelRedeem()` multiple times in a row. This way, each time the contract returns the tokens to the user, new voting units are minted to that user. The process does not revert since the contract itself is not subtracting any voting units from itself because to do this the contract has to delegate to himself.
	
	POC:
	`function test_POC_minting_infinite_votes() public {`
	`vm.startPrank(l2Bridge);`
	`drv.mint(alice, 100e18);`
	`vm.stopPrank();`
	
	`vm.startPrank(alice);`
	`drv.approve(address(stDrv), 100e18);`
	`stDrv.convert(100e18);`
	
	`assertEq(stDrv.getVotes(alice), 0);`
	`stDrv.delegate(alice);`
	`assertEq(stDrv.getVotes(alice), 100e18);`
	
	`stDrv.redeem(100e18, 16 days);`
	`stDrv.cancelRedeem(0);`
	`stDrv.redeem(100e18, 16 days);`
	`stDrv.cancelRedeem(0);`
	`stDrv.redeem(100e18, 16 days);`
	`stDrv.cancelRedeem(0);`
	`stDrv.redeem(100e18, 16 days);`
	`stDrv.cancelRedeem(0);`
	`vm.stopPrank();`
	
	`assertEq(stDrv.balanceOf(alice), 100e18);`
	`assertEq(stDrv.getVotes(alice), 500e18); // ! they should be 100e18`
	`}`

Recommendation	If `StakedDeriveToken` does not decrease the voting units of users that transfer tokens to it, then it should not increase their voting unit balance as well. This way, the functionality will not be able to be exploited for the minting of voting units. However, when the contract burns tokens by using `finalizeRedeem/deallocate`, the voting units of the user are not reduced because the tokens are being held by the contract at that time. So when tokens in the contract are burned, the voting units of their owner/ex-owner should be reduced as well.
Resolution	

Issue #05	Precision loss due to ratio being in 0 decimals
Severity	MEDIUM SEVERITY

Description	`getDeriveByVestingDuration()` determines how many `Drv` tokens the user should receive on `finalizeRedeem()`. The default redeem ratios are between 0 and 100, meaning they are not scaled to a value with higher precision like 0 - 1e18. This leads to precision loss for the users and they will receive less Drv and allocation for their stDrv.
	
	Example:
	
	- `minRedeemRatio: 50`
	- `maxRedeemRatio: 100`
	- `minRedeemDuration: 15 days`
	- `maxRedeemDuration: 90 days`
	- `duration: 82 days`
	- `stDeriveAmount: 10_000e18`
	
	`uint ratio = minRedeemRatio + ((duration - minRedeemDuration) * (maxRedeemRatio - minRedeemRatio) / (maxRedeemDuration - minRedeemDuration));`
	
	`ratio = 50 + ((82 days - 15 days) * (100 - 50) / (90 days - 15 days))`
	`ratio = 50 + (67 days * 50 / 75 days)`
	`ratio = 50 + (3350 days / 75 days)`
	`ratio = 50 + 44.6666666667 ← precision loss due to solidity not supporting floating point numbers`
	`ratio = 50 + 44`
	`ratio = 94`
	
	`10_000e18 * 94 / 100 = 9400e18`
	`10_000e18 * 94.6666666667 / 100 = 9466.6667e18`
	
	Loss for user: `9466.6667e18 - 9400e18 = 66.6667e18`

Recommendation	Consider scaling redeem related parameters and calculations to 1e18 for minimal precision loss.
Resolution	ACKNOWLEDGED The team stated this is intended.

Issue #06	`updateDividendsAddress` could result in loss of accrued dividends
Severity	LOW SEVERITY
Description	The owner has the possibility to change the `dividendsAddress` via the `updateDividendsAddress` which is the address that gets allocated automatically when `redeem` is called with a `duration > 0`. After a change, the user can call `updateRedeemDividendsAddress` to update the address of a specific redeem. The problem resides in the fact that this address can be set to address 0, if that is happening then all the old dividends for the existing redeems will not be able to automatically deallocate. In theory changing the allocation address to `address(0)` might mean the de-activation of the dividends which does not happen automatically. Furthermore, in between the change from old to new dividends address the user will accrue the rewards on the old dividends but not on the new one. This can be confusing for the users as one can think that it will automatically accrue dividends on the new address.
Recommendation	Consider changing the dividends address only for emergencies and announcing it with enough time in advance.
Resolution	ACKNOWLEDGED The team stated this is intended.

Issue #07	Approved usage ambiguous functionality
Severity	LOW SEVERITY
Description	A user can approve a certain contract to allocate/deallocate stDRV for them. The functionality is a bit confusing. First, double spending can happen—if a user wishes to decrease the approved amount with `approveUsage()`, the usage can frontrun the transaction, spend the current allowance, get assigned a new allowance and spend the new allowance. If a user wishes to decrease the allowance from 100 to 50, the usage can spend up to 100 + 50 = 150. Another issue is that for a user to allocate their own `stDRV` to a specific contract, they also has to approve that contract so they can allocate the funds. **Example:** Alice wants to allocate `100e18 stDRV` to contract X. In order for her to do that, she has to approve contract X.
Recommendation	It is better to have two methods, `increaseApproveUsage()` and `decreaseApproveUsage()`, that increase/decrease the allowance instead of assigning it directly.
Resolution	ACKNOWLEDGED The team stated this is intended.

Issue #08	`redeem` can have a duration greater than `maxRedeemDuration`
Severity	LOW SEVERITY

Description	If a redeem with a duration greater than the maxRedeemDuration is triggered by the user, the ratio will be considered as the maximum redeem ratio.
	
	The issue resides in the fact that in theory, the duration will be capped to the max redeem duration but the duration will actually not be changed and will be greater than the maximum allowed duration.
	
	Within `getDeriveByVestingDuration`:
	`// capped to maxRedeemDuration`
	`if (duration > maxRedeemDuration) {`
	`  return amount * maxRedeemRatio / 100;`
	`}`
	
	However, the duration that is saved in the `redeem` in `userRedeems` is the `duration` parameter.

Recommendation	Consider not allowing the duration to be greater than `maxRedeemDuration`.
Resolution	RESOLVED

Issue #09	The default values for several variables will not be applicable
Severity	LOW SEVERITY
Description	The contract uses OpenZeppelin's UUPS proxy pattern, meaning that `StakeDeriveToken` is the implementation contract while the proxy holds the actual state. When using a proxy, variables initialized directly at the point of declaration will not work as expected. This is because such initialization takes place in the storage of the implementation contract, not in the proxy contract's storage. To ensure proper initialization, variables should be initialized inside the initializer function, which is executed when deploying the proxy contract. Constants and immutable variables are the exception, as their values are baked into the contract's bytecode and will behave as expected, regardless of the proxy. The following variables need to be initialized correctly via the initializer and cannot be set directly at declaration: - `minRedeemRatio` - `maxRedeemRatio` - `minRedeemDuration` - `maxRedeemDuration` - `redeemDividendsAdjustment`
Recommendation	Consider initializing this in `initialize` function.
Resolution	RESOLVED

Issue #10	Implementation getter not available
Severity	INFORMATIONAL
Description	As the contracts use `UUPSUpgradable`, the implementation should have a public getter so that one can check the actual implementation. This is not present in the contract.
Recommendation	Consider adding an external function to get the implementation and that uses `ERC1967Utils.getImplementation()` to return the actual implementation.
Resolution	ACKNOWLEDGED UUPS implementation has been replaced by Transparent Proxy pattern.

Issue #11	Typographical issues
Severity	INFORMATIONAL
Description	`"redeem: stDeriveAmtount cannot be null"` *`stDeriveAmtount`* should be *`stDeriveAmount`*. — The `to` parameter of the `Convert` event can be indexed. — `__ERC20Votes_init` should also be called even if it does not do anything. — Consider following the Solidity style guide and reorder the contract layout (methods, events, state variables etc.). — Update the NatSpec comments of all functions to include params definitions: NatSpec Documentation — On line 495, consider replacing "Any vesting check should be ran before calling this" with "Any vesting check should be run before calling this". — On line 387, replace "was to be migrated" with "is to be migrated". — On line 409, replace "Cancels an ongoing redeem entry" with "redemption entry".
Recommendation	Consider fixing the typographical issues.
Resolution	PARTIALLY RESOLVED

Issue #12	Gas optimizations
Severity	INFORMATIONAL
Description	`_deleteRedeemEntry` calls `userRedeems[msg.sender]` many times; it can be cached in memory to save gas. — Unnecessary transfer of `stDRV` to `address(this)` then burn if duration is 0 in `redeem`. Consider transferring in only if `duration` is greater than 0. `_transfer(msg.sender, address(this), stDeriveAmount);` If `duration` is 0 then consider replacing `_finalizeRedeem` with a `burn` from the user directly and the additional logic. — Consider replacing all `require` statements with `if` statements and custom errors to avoid expensive string defined errors. — Within `getUserRedeem()`, `finalizeRedeem()` and `cancelRedeem()`, define `_redeem` as `memory` instead of `storage`. — Within `updateRedeemDividendsAddress()`, define `_redeem` as `memory`, and at the end of the function body replace "`_redeem.dividendsAddress = dividendsAddress;`" with "`userRedeems[msg.sender][redeemIndex].dividendsAddress = dividendsAddress;`" — Within `_allocate()`, `_deallocate()` and `cancelRedeem()`, we can remove the storage reference of `balance` and use `stDeriveBalances[msg.sender]` directly.
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	PARTIALLY RESOLVED


PALADIN
BLOCKCHAIN SECURITY