PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Preliminary Report

For Deus Finance

12 December 2023

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 full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Deus Finance on the Arbitrum 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	Deus Finance
URL	https://deus.finance/
Network	Arbitrum
Language	Solidity
Preliminary Contract	https://github.com/deusfinance/symm-migration/blob/a86859b87a33cfe8ff29ba38417b580e7fccf351
Resolution	

1.2 Contracts Assessed

Name	Contract	Live Code Match
`ClaimDeus`		
`Migrator`		

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
Governance	0	-	-	-
High	0	-	-	-
Medium	0	-	-	-
Low	10			
Informational	5			
Total	**15**	**0**	-	**0**

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 Global Issues

ID	Severity	Summary	Status
01	LOW	DEUS tokens may become temporarily locked in contracts if vote tracking is turned on	
02	LOW	Second preimage attack	
03	LOW	Initializers could be front-run	
04	INFO	Typographical issues	

1.3.2 ClaimDeus

ID	Severity	Summary	Status
05	LOW	The `SETTER_ROLE` can drain all `DEUS` tokens from the contract	
06	INFO	Gas optimizations	
07	INFO	Typographical issues	

1.3.3 Migrator

ID	Severity	Summary	Status
08	LOW	Migrations array indexes are not sufficient unique identifiers	
09	LOW	`wipeMigrations` can be front-run	
10	LOW	The setter can steal all `DEUS` tokens	
11	LOW	Users may bypass the maximum amount limit when the `bDei` contract address is changed	
12	LOW	The withdrawer may eventually withdraw funds that belong to users whose migrations have not completed yet	
13	LOW	Users can DoS wiping their own and other's migrations	
14	INFO	Missing tokens whitelist validation	
15	INFO	Typographical issues	

2 Findings

2.1 Global Issues

The issues in this section apply to the protocol as a whole. We have consolidated the global issues to simplify the report.

2.1.1 Issues & Recommendations

Issue #01	**DEUS tokens may become temporarily locked in contracts if vote tracking is turned on**
Severity	LOW SEVERITY

Description	Both of the contracts within scope interact with the DEUS contract by transferring out deposited tokens upon claim in `ClaimDeus` or direct migration/conversion in the Migrator.
	
	Within the `DEUS.transfer` and `transferFrom` functions, there is a `trackingVotes` boolean flag used to determine the execution flow:
	
	`if (trackingVotes) {`
	`  // Transfer votes`
	`  trackVotes(_msgSender(), recipient, uint96(amount));`
	`}`
	
	`_transfer(_msgSender(), recipient, amount);`
	
	If this is set to `true`, the contract will attempt to subtract the transferred amount from not only the token balance of the sender, but also from the amount of votes they have:
	
	`uint32 srcRepNum = numCheckpoints[srcRep];`
	`uint96 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;`
	`uint96 srcRepNew = sub96(`
	`  srcRepOld,`
	`  amount,`
	`  "DEUS::_moveVotes: vote amount underflows"`
	`);`

	Given this context, consider the following scenario: 1. `trackingVotes` is set to `false`. 2. `DEPOSITOR_ROLE` deposits `DEUS` tokens into the `ClaimDeus` or `Migrator` contract. 3. The admin of the `DEUS` contract sets `trackingVotes` to `true`. 4. A user tries to claim `DEUS` tokens or migrate directly from `bDEI`, `legacyDEI` or `xDEUS`. 5. The transaction reverts since the contract (`ClaimDeus` or `Migrator`) has no votes and therefore the subtraction shown above will fail due to arithmetic underflow.
Recommendation	The fix for this issue is not trivial since the code causing the actual issue seems to be in an external contract that is not within the scope of this audit. If the boolean flag is ever set to `true`, users should expect that the `claim` and `convert` functionalities may not work as intended. Do note that for code changes in the resolution round that may have an impact on the rest of the protocol, a revalidation fee may apply.
Resolution	

Issue #02	Second preimage attack
Severity	LOW SEVERITY
Description	The contracts use Merkle trees to allow specific sets of users to claim DEUS tokens. A well-known vulnerability when working with Merkle trees is the second preimage attack, which allows an adversary to use a pair of internal nodes as the value used for creating the leaf hash if the leaf has a length of 64 bytes. A warning for this same risk can be found in the OpenZeppelin's MerkleProof library at Line 13: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/3e6c86392c97fbc30d3d20a378a6f58beba08eba/contracts/utils/cryptography/MerkleProof.sol However, since the values used to create the leafs here are the claimer address and the amount to claim, the likelihood of finding appropriate internal node values is extremely low.
Recommendation	To prevent this attack surface completely, consider hashing the claimer address and corresponding maximum claim amount twice as OpenZeppelin does in their JavaScript library. Alternatively, consider adding a third element to the hashed parameters in order to make the length != 64.
Resolution	

Issue #03	Initializers could be front-run
Severity	LOW SEVERITY

Description	Both contracts are implementations of transparent upgradeable proxies deployed in a two-step manner by first deploying the actual proxy contract and then calling the initialize method.
	
	A potential attack vector is that an adversary can front-run the call to `Migrator.initialize` and call `Migrator.deposit` right after the deployment of the proxy, but just before its initialization. That way, the attacker will add their deposited tokens to the `totalLateMigratedAmount` mapping instead of `totalEarlyMigratedAmount` since the `earlyMigrationDeadline` variable will not be initialized yet and therefore will have a value of 0:
	`// @audit earlyMigrationDeadline is still 0`
	`// since the proxy initialization was front-run`
	`if (block.timestamp < earlyMigrationDeadline) {`
	`totalEarlyMigratedAmount[migrationPreferences[i]][`
	`tokens[i]`
	`] += amounts[i];`
	`} else {`
	`totalLateMigratedAmount[migrationPreferences[i]][`
	`tokens[i]`
	`] += amounts[i];`
	`}`
	
	As a result, if this depositor withdraws and another early depositor wishes to undo their migration as well, a revert could happen due to arithmetic underflow since the mappings have been incorrectly updated. Since the likelihood of this attack is very low, the overall severity is rated low.

Recommendation	Consider executing the deployment and initialization of the proxy in a single batch transaction.
Resolution	

Issue #04	Typographical issues
Severity	INFORMATIONAL
Description	Both contracts implement a `pause` function that executes the internal `_pause` method inherited from OpenZeppelin's `PausableUpgradeable` contract. Since `_pause` already applies the `whenNotPaused` modifier, there is no need to apply it a second time. Consider removing it from `ClaimDeus.pause` and `Migrator.pause`. — There is no need to explicitly inherit OpenZeppelin's `Initializable` contract since `PausableUpgradeable` and `AccessControlEnumerableUpgradeable` already inherit it and therefore so do the `ClaimDeus` and `Migrator` contracts. — The contracts are missing a call to the `_disableInitializers` method in their constructors. This is used to prevent adversaries from initializing the proxy's implementation contract directly which has led to critical vulnerabilities in the past.
Recommendation	Consider fixing the typographical issues.
Resolution	

2.2 ClaimDeus

`ClaimDeus` is an upgradeable contract used to distribute `DEUS` tokens to recipients determined by a Merkle tree. It allows privileged accounts with the `DEPOSITOR_ROLE` to deposit `DEUS` tokens into the contract. Accounts that are assigned the `SETTER_ROLE` can determine how the deposited `DEUS` tokens are distributed among users by setting the Merkle tree root.

Additionally, the contract implements `PAUSER_ROLE` and `UNPAUSER_ROLE`, which can toggle the `deposit` and `claim` functionalities.

2.2.1 Privileged Functions

- `grantRole [onlyRole(getRoleAdmin(role))]`
- `revokeRole [onlyRole(getRoleAdmin(role))]`
- `pause [onlyRole(PAUSER_ROLE)]`
- `unpause [onlyRole(UNPAUSER_ROLE)]`
- `setMerkleRoot [onlyRole(SETTER_ROLE)]`
- `deposit [onlyRole(DEPOSITOR_ROLE)]`

2.2.2 Issues & Recommendations

Issue #05	The SETTER_ROLE can drain all DEUS tokens from the contract
Severity	LOW SEVERITY
Description	The setter address is trusted to set the Merkle tree root in order to distribute the deposited DEUS tokens accordingly among stakeholders. However, this means that the setter can also set a malicious Merkle tree root that allows them to withdraw all DEUS tokens deposited and dedicated for stakeholders and migrators.
Recommendation	Consider using a strong and diverse multi-signature contract with KYC-ed or doxx-ed participants as the setter (and any other privileged account), ensuring best practices. This will reduce the risk of stolen funds in the scenario of a compromised participation key.
Resolution	

Issue #06	Gas optimizations
Severity	INFORMATIONAL
Description	The `DEUS` variable can be made constant or immutable since the value is known before deployment and such a change would save a significant amount of gas. — The proof array can be moved from `memory` to `calldata` in the claim `function`. Note that if this change is implemented, `MerkleProof.verifyCalldata` will also have to be used instead of `MerkleProof.verify` to save gas from not copying the array to memory.
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	

Issue #07	Typographical issues
Severity	INFORMATIONAL

Description	L87
	`if(claimableAmount > deusBalance) revert NotEnoughDEUS();`
	
	The underflow check is unnecessary since the subtraction on the next lines will make it implicit since the contract is compiled with a pragma version higher than 0.8.0.
	
	—
	
	A check can be added to ensure that the `claimableAmount` transferred to the user at the end of `claim` is greater than 0 to avoid having redundant/spam events emitted by adversaries claiming 0 DEUS.
	
	—
	
	The `SetMerkleRoot` event is not emitted in the `initialize` function.
	
	—
	
	L91-93
	`IERC20Upgradeable(DEUS).safeTransfer(msg.sender, claimableAmount);`
	`deusBalance -= claimableAmount;`
	`totalClaimedDeus += claimableAmount;`
	
	Consider moving the DEUS transfer after all state changes are made and before the event emission in order to correctly follow the Checks-Effects-Interactions pattern.

Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

2.3 Migrator

`Migrator` is an upgradeable contract that allows stakeholders within the DEUS ecosystem to migrate their protocol tokens to `DEUS` and `SYMMIO`. Users can create migrations to the aforementioned projects with the following options: `DEUS`, `SYMM`, and `BALANCED`. The latter allows users to choose both projects at a desired ratio.

Migrations are separated into two groups based on the time they were initiated: early and late migrations. The early migration deadline is set to 30 days after the contract's initialization.

The contract allows migrators to split, transfer, and undo (withdraw) their migrations, as well as to change their preference regarding the project they want to migrate to.

Additionally, three functionalities allow for direct conversions from `bDEI`, `legacyDEI`, and `xDeus` tokens to `DEUS` at fixed rates.

The admin of the contract can wipe migration records from the contract storage once a migration has been completed. There is also a `WITHDRAWAL_ROLE` which allows privileged addresses to withdraw any funds (ERC20 tokens) from the contract.

Accounts that are assigned the `SETTER_ROLE` can update the Merkle tree root, which determines the maximum amount of `bDEI` and `legacyDEI` tokens a user is allowed to exchange (migrate). They can also update the `bDEI` token address.

Similar to the `ClaimDeus` contract, `Migrator` implements `PAUSER_ROLE` and `UNPAUSER_ROLE` that can toggle available functionalities.

2.3.1 Privileged Functions

- `grantRole [onlyRole(getRoleAdmin(role))]`
- `revokeRole [onlyRole(getRoleAdmin(role))]`
- `pause [onlyRole(PAUSER_ROLE)]`
- `unpause [onlyRole(UNPAUSER_ROLE)]`
- `setMerkleRoots [onlyRole(SETTER_ROLE)]`
- `setBDEIAddress [onlyRole(SETTER_ROLE)]`
- `withdraw [onlyRole(WITHDRAWER_ROLE)]`
- `wipeMigrations [onlyRole(DEFAULT_ADMIN_ROLE)]`

2.3.2 Issues & Recommendations

Issue #08	Migrations array indexes are not sufficient unique identifiers
Severity	LOW SEVERITY

Description	Each user's migrations are accessed via their indexes in the `migrations[user]` array:
	
	`// list of user migrations`
	`mapping(address => Migration[]) public migrations;`
	
	Indexes are currently used as IDs for all migrations. However, this is not a sufficient unique identifier for each record record since upon transfers, splits and removals (undos), the indexes of the elements change due to the changed order and swaps. This can lead to race conditions resulting in unexpected behavior and situations for the users.
	
	Consider the following scenario:
	
	1. A user splits their migration of 100 tokens (the only one they have, so the index is #0).
	2. Then they undo (withdraw) the first portion of the split migration and keep the latter.
	3. They call `split(0, 90)` and then `undo(0)`.
	4. The expected outcome is that the user would be left with 1 migration of 90 tokens (the second part of the split is temporarily at index #1).
	5. However, the user did not wait for the first transaction to be finalized, so both transactions/calls are currently in the mempool.
	6. For some reason (e.g. higher gas fee), the second transaction gets executed first.
	7. The whole migration of 100 tokens is now removed and withdrawn instead of just the intended portion of 10 tokens (90 should have been kept in the `Migrator`).

Recommendation	Consider assigning a unique identifier to each migration record, for example, based on a hash of all properties to prevent the above scenarios from happening.
Resolution	

Issue #09	**`wipeMigrations` can be front-run**
Severity	LOW SEVERITY
Description	`wipeMigrations` is used to delete users' migrations records from storage for given tokens once the migrations have been completed. However, an adversary can monitor the mempool and execute `transfer`, `split` or undo actions which will prevent the admin from wiping their migrations. Additionally, if a user has created a migration just shortly before the admin calls `wipeMigrations`, it can be unintentionally wiped even if not completed yet.
Recommendation	Consider if this would become an issue. A possible solution is to add the `whenPaused` modifier to the `wipeMigrations` function to make sure users are not able to front-run and split, transfer and undo their migrations. Also, consider passing the specific identifiers for all migrations (currently array indexes) that have to be deleted instead of using token addresses as a filter.
Resolution	

Issue #10	The setter can steal all DEUS tokens
Severity	LOW SEVERITY
Description	Similar to the issue in `ClaimDeus`, the `SETTER_ROLE` is responsible for setting the Merkle tree roots that are used to determine which users can exchange their `bDei` and `legacyDei` tokens at a fixed rate for `DEUS`. The setter account can also change the `bDei` address, which means they can insert a mock contract that deposits 0 actual tokens but still exchanges them for as many `DEUS` as desired.
Recommendation	Consider making the `bDei` contract address a constant as well as using a multi-signature contract.
Resolution	

Issue #11	**Users may bypass the maximum amount limit when the bDei contract address is changed**
Severity	LOW SEVERITY

Description	The bDei token contract address can be changed by privileged SETTER_ROLE accounts.
	
	The amount of bDei tokens converted to DEUS is tracked in the convertedAmount mapping as below:
	
	`convertedAmount[msg.sender][bDEI] += amount;`
	`require(convertedAmount[msg.sender][bDEI] <= maxAmount, "Amount Too High");`
	
	When the value of bDei is changed, the mapping will be reset as the converted amount of the new token contract will be 0. Therefore, if the bDEIMerkleRoot is not updated accordingly for the new token, users will be able to reuse the allowances from the old/current Merkle root and convert another maxAmount of the new bDEI token to DEUS:
	`if (`
	`!MerkleProof.verify(`
	`proof,`
	`bDEIMerkleRoot,`
	`keccak256(abi.encode(msg.sender, maxAmount))`
	`)`
	`) revert InvalidProof();`

Recommendation	Consider making the bDei token contract address a constant. Alternatively, include the address of the bDei token in the leaf so that the old bDEIMerkleRoot is not reusable.
Resolution	

Issue #12	**The withdrawer may eventually withdraw funds that belong to users whose migrations have not completed yet**
Severity	LOW SEVERITY

Location	`function withdraw(`
	`address[] memory tokens`
	`) external onlyRole(WITHDRAWER_ROLE) {`
	`for (uint256 i; i < tokens.length; ++i) {`
	`IERC20Upgradeable(tokens[i]).safeTransfer(`
	`msg.sender,`
	`IERC20Upgradeable(tokens[i]).balanceOf(address(this))`
	`);`
	`}`
	`}`

Description	The withdrawer can withdraw the whole balance of the `Migrator` contract of given tokens. However, since there is no validation for the amount to withdraw or the time when the funds are withdrawn, a migrator who has not yet completed their migration and wants to undo it will not be able to do so since their tokens might have already been withdrawn by the withdrawer and therefore calling undo will revert.
Recommendation	Consider whether this could become an issue and implement a more efficient solution that keeps track of users' deposits and completed and pending migrations. Do note that for logic changes in the resolution round that may have an impact on the rest of the protocol, a revalidation fee may apply.
Resolution	

Issue #13	Users can DoS wiping their own and other's migrations
Severity	LOW SEVERITY
Description	A user can create a migration with any token and any amount on behalf of any account. A potential attack vector that can result in the `getMigrations` and `wipeMigrations` functions getting DoS-ed is where a malicious user creates a huge amount of migrations with just 1 wei of deposit or even 0. This can be done to both prevent the protocol admin from wiping your own migrations or any others.
Recommendation	As described in the issue below, consider implementing a token whitelist and add a minimum (so-called "dust") amount of tokens to be migrated. Keep in mind that this validation should also be applied consistently in the split and transfer logic.
Resolution	

Issue #14	Missing tokens whitelist validation
Severity	INFORMATIONAL
Description	Users are currently able to create migrations for any `ERC20` tokens or even mock smart contracts since there is no input validation applied on the passed addresses. This can be used to grief other users. Some users can also unintentionally lose their funds if they use non-standard ERC20 tokens listed in the following repository: https://github.com/d-xo/weird-erc20. Also, if a user deposits `DEUS` tokens to create a migration, those may be withdrawn by other users who exchange `xDeus` directly for `DEUS` and at some point use the depositors' `DEUS` tokens instead of the protocol-supplied ones.
Recommendation	Consider implementing a token whitelist that will allow a privileged address to determine which tokens can be used for creating migrations.
Resolution	

Issue #15	Typographical issues
Severity	INFORMATIONAL
Description	A zero-array-length check can be added in the `deposit` function to prevent spamming meaningless events. Also, consider adding a check to verify that the lengths of both arrays is the same. — A zero-amount check can be added at the end of each `convert` function to make sure that no 0 DEUS amounts are transferred. — The checks made on lines 217, 221, 240, 304 are redundant since both underflow and index-out-of-bound checks are done implicitly when a subtraction is made or an array element is accessed respectively. — The addresses of the `legacyDEI` and `xDeus` token contracts are hardcoded as local variables into the corresponding `convert` functions. Moving them to a `constant` or `immutable` variable would be more appropriate for the case. — `setBDEIAddress`, `withdraw` and `wipeMigrations` do not emit events while they perform critical state changes. Consider emitting corresponding events.
Recommendation	Consider fixing the typographical issues.
Resolution	


PALADIN
BLOCKCHAIN SECURITY