PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Cian (Polygon)

03 January 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 Cian's contracts on the Polygon 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.

This audit is an extension from the Ethereum audit. All acknowledged issues remain valid in this audit.

1.1 Summary

Project Name	Cian
URL	https://cian.app/
Network	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
AdapterBase	Dependency	MATCH
OneInchAdapter	0x9633D6C81E9449B05954B74c257F5964B6864cAA	MATCH
AaveV3Adapter	0x67709Ce1908077801567998a23Ab3ce10C45727D	MATCH
BalancerV2Adapter	0x8ebdc47aE80f411b8722E1aCe00DcE28a38Cf273	MATCH
FeeBoxMATIC	0x1C8126e02e8A7dAc69FD6444Ef0b8be5430DF776	MATCH
QuickSwapAdapter	0x6C5766Bd236BF879dF4FF468740A8E3FB0Ac12D3	MATCH
StaderAdapter	0x4E231b636e799d19a54065Ba79A67D8aFA1dDFa0	MATCH
WmaticGateway	0xdCB3D91555385DaE23e6B966b5626aa7A75Be940	MATCH
AdapterManager	0x907883da917ca9750ad202ff6395C4C6aB14e60E	MATCH
AccountManager	Not deployed	N/A
Automation	0xA79D00C0feA6bAABE8A1fEd0c41C4d36E7B81895	MATCH
AutomationCallable	Dependency	MATCH
ControllerLib	0xff6771a9565F18638faB2972BA7Fc798ad8bCad0	MATCH
ControllerLibSub	0xEa5f10A0E612316A47123D818E2b597437D19a17	MATCH
ControllerLink	0x6E3066412B4e67d2933d6023a7c58d63DD8f800a	MATCH
BalancerERC3156 (V2)	0xf1a5710a91183e317b17d1A314227B36d1a30b95	MATCH
ERC2612Verifier	0xE946Dd7d03F6F5C440F68c84808Ca88d26475FC5	MATCH
TokenApprovalVerifier	0x9B2316cfe980515de7430F1c4E831B89a5921137	MATCH
StaderAirdrop	0x406e1e0e3cb4201B4AEe409Ad2f6Cd56d3242De7	MATCH
Timelock	0xCe672de0D2d38944716c21BCA7DB1164685Af2aC	MATCH
TimelockCallable	Dependency	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1	-	-
Medium	4	2	1	1
Low	7	4	-	3
Informational	9	2	-	7
Total	**21**	**9**	**1**	**11**

Classification of Issues

Severity	Description
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	INFO	Typographical errors	ACKNOWLEDGED

1.3.2 AdapterBase

ID	Severity	Summary	Status
02	MEDIUM	Adapters will not fail if a wrong function is called	PARTIAL

1.3.3 OneInchAdapter

No issues found.

1.3.4 AaveV3Adapter

ID	Severity	Summary	Status
03	LOW	Matic can get stuck in the contract during a deposit	✓ RESOLVED
04	LOW	Withdrawals can be blocked if a token is removed from the list of trusted tokens	ACKNOWLEDGED
05	INFO	Lack of events for the `initialize` function	ACKNOWLEDGED
06	INFO	`AaveRepay` event might emit the wrong parameter	✓ RESOLVED
07	INFO	Unused import: IAToken.sol	ACKNOWLEDGED

1.3.5 BalancerV2Adapter

ID	Severity	Summary	Status
08	HIGH	The adapter does not return leftover balances	✓ RESOLVED
09	MEDIUM	The adapter is not compatible with tokens that have a fee on transfer	✓ RESOLVED

1.3.6 FeeBoxMatic

No issues found.

1.3.7 VerifierBasic

No issues found.

1.3.8 QuickSwapAdapter

ID	Severity	Summary	Status
10	MEDIUM	The adapter is not compatible with tokens that have a fee on transfer	✓ RESOLVED
11	INFO	router can be made constant	ACKNOWLEDGED

1.3.9 StaderAdapter

ID	Severity	Summary	Status
12	LOW	No view function for pending requests	✓ RESOLVED
13	INFO	Unused variable	ACKNOWLEDGED
14	INFO	`payable` keyword not needed for stake	ACKNOWLEDGED

1.3.10 WmaticGateway

No issues found.

1.3.11 AdapterManager

No issues found.

1.3.12 AccountManager

ID	Severity	Summary	Status
15	LOW	Authorized addresses are difficult to query	ACKNOWLEDGED

1.3.13 Automation

No issues found.

1.3.14 AutomationCallable

No issues found.

1.3.15 ControllerLib

ID	Severity	Summary	Status
16	MEDIUM	Privilege escalation: The approvals functions allow the `advancedTradingEnable` boolean to be bypassed	ACKNOWLEDGED

1.3.16 ControllerLibSub

No issues found.

1.3.17 ControllerLink

No issues found.

1.3.18 BalancerERC3156 (V2)

ID	Severity	Summary	Status
17	LOW	`maxFlashLoan` returns an incorrect value	✓ RESOLVED
18	LOW	The reentrancy check is flawed	✓ RESOLVED
19	INFO	`vault` can be made constant	✓ RESOLVED

1.3.19 ERC2612Verifier

No issues found.

1.3.20 TokenApprovalVerifier

No issues found.

1.3.21 StaderAirdrop

ID	Severity	Summary	Status
20	LOW	Rewards can be distributed twice if the contract is paused too late	ACKNOWLEDGED
21	INFO	Typographical error	ACKNOWLEDGED

1.3.22 Timelock

No issues found.

1.3.23 TimelockCallable

No issues found.

2 Findings

2.1 Global Issues

The issues in this section occur across multiple contracts within the protocol.

2.1.1 Issues & Recommendations

Issue #01	Typographical errors
Severity	INFORMATIONAL

Description	We have consolidated the typographical errors into a single issue to keep the report brief and readable.
	
	`ProxyWallet::40 (example for variables)`
	`address public immutable userDatabase;`
	
	`ProxyWallet::76 (example for parameters)`
	`function proxyAdminCheck(address defaultProxyAdmin)`
	
	Throughout the codebase, tokens and other contracts are almost never cast to their correct type. This requires the developer to then explicitly cast them to `IERC20`, `IControllerLink`, `IAdapterManager`, etc. The developer should consider always immediately specifying the types as the correct types instead of using the generic "`address`" type. Although this will not affect gas usage, it heavily simplifies the codebase and also indicates to third parties that the developer has a good understanding of solidity best practice.
	
	`pragma solidity >=0.8.0 <0.9.0;`
	
	This can be simplified to `pragma solidity ^0.8.0` which restricts the version to 0.8 compatible versions as well.

Recommendation	Consider fixing the typographical errors.
Resolution	ACKNOWLEDGED

2.2 Adapters/AdapterBase

This is the code for the `AdapterBase` contract, which is an abstract contract that defines a basic adapter template. The contract is `Ownable`, which means that it has an owner address that can be used to control access to the contract's functions. The contract is also `TimelockCallable`, which means that it can be called by a `Timelock` contract.

The contract has a `constructor` function that takes an adapter manager address, a timelock address, and a name for the adapter as input. The contract also has functions for pulling tokens from an address, approving tokens, returning assets to an address, and sweeping assets from an address.

Note that the privileged functions are present in all adapters and will not be repeated in the following adapter sections.

No significant changes were made since the Avalanche audit. Acknowledged issues from the previous audit are not listed again (as goes for all contracts within this audit).

2.2.1 Privileged Functions

- `sweep [ timelock ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`
- `setTimelock [ timelock ]`

2.2.2 Issues & Recommendations

Issue #02	Adapters will not fail if a wrong function is called
Severity	MEDIUM SEVERITY
Description	`AdapterBase` is inherited by all adapters and defines an empty `fallback` and an empty `receive` function. This means that any adapter can receive Ether directly, and if a function is called that was not defined in the adapter, the transaction would not fail. For example, if someone calls an Aave function using the 1inch Adapter, the function will not revert. This issue becomes annoying when calling multiple adapters at a time because you would not know which call did nothing. Additionally, any adapter can receive Matic directly even if they should not ever receive Matic directly.
Recommendation	Consider removing the `fallback`/`receive` functions and defining it only within the adapter that needs them.
Resolution	PARTIALLY RESOLVED The `fallback` function was removed, but the `receive` function was kept.

2.3 Adapters/OneInchAdapter

`OneInchAdapter` inherits from the `AdapterBase` contract and allows for automation to use 1inch to swap for a wallet.

The contract also defines a public constant for the `oneInchRouter` which is hard-coded to be the address `0x1111111254fb6c44bAC0beD2854e76F90643097d`.

Finally, the contract defines a function called `swap` which can be called via delegation in order to perform a swap of tokens using the `OneInchRouter` contract. The function takes two arguments (a `bytes memory callArgs`, and a `uint256 amountETH`), and blindly uses these to call the `OneInchRouter` contract, requiring said call to succeed. Any function can therefore be called on the router.

`OneInchAdapter` is a `delegatecall` adapter.

2.3.1 Issues & Recommendations

No issues found.

2.4 Adapters/AaveV3Adapter

`AaveV3Adapter` allows CIAN users to interact with the AaveV3 protocol on the Polygon network. Users can deposit, withdraw, borrow, repay, claim rewards and activate `EMode`.

The following functions are meant to be called by the `AdapterManager`:

- `deposit`
- `withdraw`

The following functions are only callable via `delegatecall`:

- `setCollateral`
- `borrow`
- `approveDelegation`
- `payback`
- `claimRewards`
- `setUserEMode`

2.4.1 Privileged Functions

- `initialize [ onlyTimelock ]`

2.4.2 Issues & Recommendations

Issue #03	Matic can get stuck in the contract during a deposit
Severity	LOW SEVERITY
Description	The deposit function does not check that the `msg.value` is 0 when adding a token other than Matic. A deposit with bad parameters could lock Matic in that contract.
Recommendation	Consider checking that `msg.value` is 0 during a deposit of a token that is not Matic.
Resolution	RESOLVED The `msg.value` check was added.

Issue #04	Withdrawals can be blocked if a token is removed from the list of trusted tokens
Severity	LOW SEVERITY
Description	The `withdraw` function uses the list of trusted tokens to return the address of the `aToken`. However, if a token was once in the trusted list and subsequently removed, users will not be able to withdraw from Aave using the adapter. This issue is only rated as low as users would still be able to use Aave directly to withdraw their tokens.
Recommendation	Consider whether this is a problem, and if it is, consider fixing it.
Resolution	ACKNOWLEDGED

Issue #05	Lack of events for the `initialize` function
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the function.
Resolution	ACKNOWLEDGED

Issue #06	`AaveRepay` event might emit the wrong parameter
Severity	INFORMATIONAL

Description	Currently, the `AaveRepay` event emits the following:
	`emit AaveRepay(tokenAddr, amount, address(this), rateMode);`
	While this works for all standard operations, an incorrect parameter is emitted if `amount` is `uintMax`:
	`if (amount == type(uint256).max) {`
	`uint256 repayValue = IERC20(debtMaticAddr).balanceOf( address(this) );`
	For this case, the event should emit `repayValue` instead of amount.

Recommendation	Consider updating the `amount` value directly instead of using `repayValue`. That way, the event will be emitted with the right amount directly.
	`if (amount == type(uint256).max) {`
	`amount = IERC20(debtMaticAddr).balanceOf( address(this) );`

Resolution	RESOLVED

Issue #07	Unused import: IAToken.sol
Severity	INFORMATIONAL
Description	Variables, functions and imports that are defined within the code but never used can be removed to make the coder cleaner and more readable.
Recommendation	Consider removing the following import: `import "../../../interfaces/aave/v2/IAToken.sol";`
Resolution	ACKNOWLEDGED

2.5 Adapters/BalancerV2Adapter

`BalancerV2Adapter` allows CIAN users to swap tokens using the Balancer protocol. The user can choose between two swap methods:

- `batchSwap`
- `singleSwap`

Essentially, users can either swap a token for another token using just one pool (`singleSwap`) or they can swap one token for another and then again for another token using multiple pools. This can also be done using only a partial amount of each index token (`batchSwap`).

2.5.1 Issues & Recommendations

Issue #08	The adapter does not return leftover balances
Severity	HIGH SEVERITY
Description	The adapter calls `pullAndApprove` with the amount `limit[0]`. This amount is meant to be the maximum amount that can be used for a swap. Consider the following scenario: 1. Alice wants to swap USDT to WETH. 2. Alice selects `kind` as `GIVEN_OUT`. 3. `GIVEN_OUT` calculates `amountIn` based on the desired output amount `poolRequest.amount`. 4. Alice uses `limit[0]` as `1400` and `singleSwap.amount` as `1`. This means Alice intends to swap her USDT to 1 WETH with a maximum spending limit of 1400 USDT. 5. The Adapter now executes a `transferFrom` of 1400 USDT from Alice and executes the swap. 6. The actual price is however 1300 USDT per WETH which results in the Balancer protocol only taking 1300 USDT from the Adapter: `_receiveAsset(singleSwap.assetIn, amountIn, funds.sender, funds.fromInternalBalance);` 7. This results in the leftover balance of 100 USDT stuck in the Adapter while only the owner can withdraw the token sent in excess, thus essentially resulting in a loss for Alice. Moreover, within `_handleRemainingEth`, the leftover Ether is sent to `msg.sender`, which is the adapter. This PoC applies to both `singleSwap` and `batchSwap`.
Recommendation	Consider calculating the difference between `input[0]` and the amount which was actually taken by the Balancer protocol, then send it back to the user at the end of the function. Additionally, `_handleRemainingEth` should send the leftover amount back to `funds.recipient` or `tx.origin`.

Resolution	RESOLVED Additionally, a non-reentrant check was also added to prevent reentrancy.

Issue #09	**The adapter is not compatible with tokens that have a fee on transfer**
Severity	MEDIUM SEVERITY
Description	The adapter is not compatible with the swapping of tokens that have a fee on transfer. This issue essentially arises when `limit[0]` is transferred via `pullAndApprove`, as the contract will receive less tokens which will eventually result in a revert within `_receiveAsset`.
Recommendation	Consider either acknowledging this issue or adding logic to support tokens with a fee on transfer.
Resolution	RESOLVED However, a before-after pattern should be added.

2.6 Adapters/FeeBoxMATIC

`FeeBoxMATIC` is responsible for taking fees from users' wallets to subsidize gas and management costs for the operators that execute automation jobs on their behalf.

All functions are meant to be called by the `AdapterManager`.

2.6.1 Privileged Functions

- `initialize [ timelock ]`
- `setAdapterManager [ timelock ]`
- `paymentCheck [ balanceController ]`
- `setBalance [ balanceController ]`

2.6.2 Issues & Recommendations

No issues found.

2.7 Adapters/VerifierBasic

`VerifierBasic` is used by the various FeeBoxes to validate signatures.

2.7.1 Issues & Recommendations

No issues found.

2.8 Adapters/QuickSwapAdapter

`QuickSwapAdapter` allows CIAN users to swap tokens using the QuickSwap decentralized exchange (DEX). Users can execute the following swaps:

- ETH -> exact tokens
- exact ETH -> tokens
- tokens -> exact tokens
- exact tokens -> tokens
- tokens -> exact ETH
- exact tokens -> ETH

`QuickSwapAdapter` is solely meant to be called by the user via the `AdapterManager` contract.

2.8.1 Issues & Recommendations

Issue #10	**The adapter is not compatible with tokens that have a fee on transfer**
Severity	MEDIUM SEVERITY
Description	Currently, the adapter is not compatible with swapping tokens that have a fee on transfer.
Recommendation	Consider acknowledging this issue if support for such tokens is not intended, otherwise consider implementing all swap functions to support such tokens. Moreover, if these functions are implemented, the logic for `pullTokensIfNeeded` must be adjusted because the contract will receive fewer tokens, and therefore, the transfer from the adapter to the pair will revert.
Resolution	RESOLVED

Issue #11	**`router` can be made constant**
Severity	INFORMATIONAL
Location	L10 `IQuickSwapRouter internal router = IQuickSwapRouter(routerAddr);`
Description	Variables that are never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly constant to save gas.
Resolution	ACKNOWLEDGED

2.9 Adapters/StaderAdapter

`StaderAdapter` allows CIAN users to interact with the Stader Labs protocol under the following address: `0xfd225C9e6601C9d38d8F98d8731BF59eFcF8C0E3`. `StaderAdapter` is a pure `delegatecall` adapter meant to be called by the users' proxy.

Users can:

1. `stake`, which means swapping `MATIC` for `maticX`.
2. `requestUnstake`, which swaps `maticX` to `MATIC` and prepares the corresponding `MATIC` to be claimed by the user after a lockup.
3. `claimUnlocked`, which simply claims the requested position after the lockup period.

2.9.1 Issues & Recommendations

Issue #12	No `view` function for pending requests
Severity	LOW SEVERITY
Description	Currently, the adapter only has the function `claimUnlocked` which claims the position based on the `index` parameter. However, there is no way for the user to see their actual requested positions other than interacting with the Stader smart contract directly using the proxy address as a parameter.
Recommendation	Consider implementing a `view` function that fetches the data from `getUserMaticXSwapRequests`.
Resolution	RESOLVED

Issue #13	Unused variable
Severity	INFORMATIONAL
Location	L8 & L9 `address public constant maticXAddr = 0xfa68FB4628DFF1028CFEc22b4162FCcd0d45efb6;`
Description	The variable above is unused.
Recommendation	Consider removing the unused variable.
Resolution	ACKNOWLEDGED

Issue #14	`payable` keyword not needed for `stake`
Severity	INFORMATIONAL
Description	While the `stake` function indeed transfers Matic to another contract, it is only callable via a `delegatecall`. However, as a delegate call cannot have a `msg.value`, the Matic would be transferred to the same contract which would just be a waste of gas.
Recommendation	Consider removing the `payable` keyword.
Resolution	ACKNOWLEDGED

2.10 Adapters/WmaticGateway

`WmaticGateway` is a simple adapter that allows for the depositing and withdrawal of `Wmatic` from and into `Matic`.

It should be noted that withdrawing `Wmatic` straight into a proxy is generally a discouraged practice due to the fallback logic of a proxy costing potentially too much gas for the gas-limited transfer to succeed. However, as the wallet proxy presently has a `receive()` override, this should not cause a problem for now. Generally and informationally speaking, a non-upgradeable helper contract is used to withdraw WETH instead of the approach which is taken here.

`WmaticGateway` is a `delegationcall` adapter.

2.10.1 Issues & Recommendations

No issues found.

2.11 Adapters/AdapterManager

`AdapterManager` is the main registry for all Cian adapters. An adapter is a smart contract that Cian operators can use to execute functionality for users on their wallets.

The manager can also be paused by various Cian approved pause guardians. This prevents operators from executing calls on user wallets and can be used as an emergency safeguard if an adapter has a vulnerability.

2.11.1 Privileged Functions

- `execute [ user proxies ]`
- `registerAdapters [ timelock]`
- `unregisterAdapters [ timelock ]`
- `setPauseWhiteList [ timelock ]`
- `setPause [ suspend permissioned accounts & owner can pause, timelock can unpause ]`

2.11.2 Issues & Recommendations

No issues found.

2.12 Core/AccountManager

`AccountManager` is a helper contract that is deployed for each user that aims to increase the comfort when handling an arbitrary amount of `Accounts`. The owner of this contract can add various `Accounts` to the `AccountManager` and grant arbitrary addresses privileged rights to execute the following functions for a `userAccount` (`IAccount`) on the previously added `Accounts`:

- `createSubAccount`
- `executeOnAdapter`
- `executeMulticall`
- `setAdvancedOption`
- `callOnSubAccount`
- `withdrawAssets`
- `approveTokens`

It also allows privileged addresses to call `approve` on the `ERC2612Verifier` as well on the `tokenApprovalVerifier` contract.

As mentioned above, the owner of this contract has the privilege to add and delete accounts via `addAccounts` and `delAccounts`. Before any accounts can be added, the ownership of this account must be transferred to the `AccountManager`.

The most privileged function is the `setAuthorization` function which allows the owner to set any address as executor for specific operations for any account within a certain deadline.

These are the following operations that can be assigned to the executor:

- `CREATE_SUBACCOUNT`
- `EXECUTE_ON_ADAPTER`
- `MULTICALL`
- `SET_ADVANCED_OPTION`
- `CALL_ON_SUBACCOUNT`
- `WITHDRAW_ASSETS`
- `APPROVE_TOKENS`
- `APPROVE_ERC2612_VERIFIER`
- `APPROVE_TOKEN_VERIFIER`

If an address was set as executor for an account with the correct operation, it can execute the function which was assigned to the operation arbitrarily often within the determined deadline.

The owner can also freely define the `ERC2612Verifier` and the `TokenApprovalVerified` as well as change the `minDelay` and `maxDelay` which is used for granting the authorization.

2.12.1 Privileged Functions

- `transferOwnership`
- `renounceOwnership`
- `setDelay`
- `setVerifier`
- `addAccounts`
- `delAccounts`
- `setAuthorization`

2.12.2 Issues & Recommendations

Issue #15	Authorized addresses are difficult to query
Severity	LOW SEVERITY
Description	An account could forget which address they authorized. They would need to query events to get them, which is a complicated process, and not all users may be able to do that.
Recommendation	Consider adding the authorized address to a set so the user can query which address was authorized for which accounts more easily.
Resolution	ACKNOWLEDGED

2.13 Core/Automation

`Automation` is the core authorization contract used by all wallets. Operators must go through the `Automation` contract if they wish to execute automation tasks on a user wallet. `Automation` will then call the `ERC2612Verifier` to check if the operator has permission to execute the specific action for the user.

`TokenApprovalVerifier` will be queried if the action deals with tokens.

The user can also set a `LoanProvider` that will be used for flashloans, and if none are defined, the default one will be used.

2.13.1 Privileged Functions

- `setLoanProvider [ only account owner ]`
- `autoExecute [ only approved adapters ]`
- `autoExecuteMultiCall [ only approved adapters ]`
- `autoApprove [ only if 0 was approved and spender needs to have been approved ]`
- `autoApproveWithPermit [ only if 0 was approved and owner has signed a message to permit ]`
- `doFlashLoan [ only if 1 was approved ]`
- `autoExecuteOnSubAccount [ only if 2 was approved ]`
- `doFlashLoanOnSubAccount [ only if 3 was approved ]`

2.13.2 Issues & Recommendations

No issues found.

2.14 Core/AutomationCallable

`AutomationCallable` is a contract that needs to be inherited to allow the contract to set an `autoExecutor` which allows it to execute tasks on the contract.

2.14.1 Issues & Recommendations

No issues found.

2.15 Core/ControllerLib

`ControllerLib` represents the core contract of the CIAN architecture — it is the implementation of the user's `ProxyWallet`, which is their virtual wallet. `ControllerLib`, therefore, contains all core logic for the user and other system components to manage the user's virtual wallet.

It allows the user to force their virtual wallet to execute arbitrary logic through either calls or delegatecalls. It also allows the user to approve various controllers to execute logic on adapters for them. These controllers do this by calling the `CallProxy` (called the "automation" in this contract) which is also described within this audit. The `CallProxy` then validates the request and forwards it to the user's virtual wallet.

2.15.1 Privileged Functions

- `createSubAccount [ owner ]`
- `executeOnAdapter [ automation / owner ]`
- `multiCall [ automation / owner ]`
- `callDirectly [ owner ]`
- `callOnSubAccount [ automation / owner ]`
- `setAdvancedOption [ owner ]`
- `withdrawAssets [ owner ]`
- `approve [ automation / owner ]`
- `approveTokens [ automation / owner ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`
- `reinitialize [ owner ]`

2.15.2 Issues & Recommendations

Issue #16	**Privilege escalation: The approvals functions allow the `advancedTradingEnable` boolean to be bypassed**
Severity	MEDIUM SEVERITY

Location	L299–339
	`function withdrawAssets(`
	`address[] memory _tokens,`
	`address _receiver,`
	`uint256[] memory _amounts`
	`) external onlyOwner {`
	`[...]`
	`}`
	
	`function approve(`
	`IERC20 _token,`
	`address _spender,`
	`uint256 _amount`
	`) external onlyAutomationOrOwner {`
	`[...]`
	`}`
	
	`function approveTokens(`
	`IERC20[] memory _tokens,`
	`address[] memory _spenders,`
	`uint256[] memory _amounts`
	`) external onlyAutomationOrOwner {`
	`[...]`
	`}`

Description	In order to withdraw tokens to an external address, the owner needs to allow `advancedOptionEnable`. A privilege escalation can occur by approving an external address as the spender. This spender can then call `transferFrom` to withdraw the tokens. This can be done by the owner or the automation.
Recommendation	Consider whether this is an issue, and if so, consider preventing these functions from being called when that bool is set to `false`.

Resolution	ACKNOWLEDGED The team does not consider this an issue since the advanced mode is to let users execute arbitrary operations rather than withdrawing funds.

2.16 Core/ControllerLibSub

`ControllerLibSub` represents a sub-wallet of the main `ControllerLib` wallet with less strict permission controls. The main wallet has full authorization over this sub wallet as well as the main wallet owner.

Most of the issues from `ControllerLib` are present here as well.

2.16.1 Privileged Functions

- `reinitialize [ eoa owner ]`
- `withdrawAssets [ eoa owner ]`
- `approveTokens [ eoa owner ]`
- `executeOnAdapter [ owner: parent wallet ]`
- `multiCall [ owner: parent wallet ]`

2.16.2 Issues & Recommendations

No issues found.

2.17 Core/ControllerLink

`ControllerLink` is a helper contract that acts like a user database. Every time a new `ProxyWallet` is created, it is added to the `ControllerLink` mappings.

2.17.1 Privileged Functions

- `addAuth [ factory ]`
- `removeAuth [ owner ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`

2.17.1 Privileged Functions

No issues found.

2.18 Core/BalancerERC3156 (V2)

`BalancerERC3156` is a simple user interface for executing flashloans. The user can request a flashloan from the `vault` with an arbitrary `borrower` address as receiver.

The `vault` will then send the tokens to the contract and these tokens will then be sent to the `borrower` to execute its logic with the tokens.

After the logic is executed, `BalancerERC3156` will take the tokens + fee from the `borrower` and send it back to the `vault`.

2.18.1 Issues & Recommendations

Issue #17	`maxFlashLoan` returns an incorrect value
Severity	LOW SEVERITY
Description	Currently, the `maxFlashLoan` returns `uint256(max);` however, the function name indicates that the goal for this function is to return the maximum flashloan amount.
Recommendation	Consider removing this function or adding logic that returns the maximum possible amount of a flashloan for a specific token, i.e. the token's balance of the vault.
Resolution	RESOLVED

Issue #18	The reentrancy check is flawed
Severity	LOW SEVERITY
Location	L70 `require(executor != address(0), "reEntrance");`
Description	This requirement is not a reentrancy check, but it ensures that the flashloan was initiated by this contract with the `flashLoan` function. A reentrancy could in theory still be made, though we are sure the balancer implementation protects against this.
Recommendation	Consider reverting with a more accurate message. Also, if a reentrancy check was needed, there can be a check that `executor` is `address(0)` at line 50.
Resolution	RESOLVED

Issue #19	`vault` can be made `constant`
Severity	INFORMATIONAL
Description	Variables that are never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly `constant`.
Resolution	RESOLVED

2.19 Core/ERC2612Verifier

`ERC2612Verifier` allows users to specify if they approve basic operations and/or specific adapters. Those approvals are represented using ids. If a user wants to allow a specific id, they need to call approve with `2^id` as the `approvalType`.

In addition, an user can sign a message to approve an adapter without ever calling the function themselves.

Currently the basic operations are:

- (2^0): approve a token
- (2^1): allow flashloans on Balancer

The `id` of the different adapters will be chosen by the team.

Note that any approval will overwrite all previous approvals. This means that the user must be extremely careful with their transaction bytes, as it will be exceptionally difficult to figure out which adapter they are approving.

2.19.1 Privileged Functions

- `approve [only owner of that account]`
- `revoke [only owner of that account]`

2.19.1 Issues & Recommendations

No issues found.

2.20 Core/TokenApprovalVerifier

`TokenApprovalVerifier` allows users to approve different addresses to use the tokens that are in their proxies. They can also sign a message that can be used to approve on behalf of the user.

2.20.1 Privileged Functions

- `approve [ proxies owner ]`

2.20.2 Issues & Recommendations

No issues found.

2.21 Core/StaderAirdrop

`StaderAirdrop` contract is a simple airdrop contract that uses `merkleTree` cryptography to whitelist specific addresses with specific amounts without having to set amounts on-chain, which is way more expensive.

The owner simply adds addresses and amounts as leaves to the Merkle tree, calculates the root, and then calls `updateMerkleRoot` with the calculated `root`.

Any address can then claim the assigned amount on behalf of the privileged address. The contract allows for unlimited rounds, and each round changes when a new `merkleRoot` is set via `updateMerkleRoot`. Each address can only be claimed once per round.

2.21.1 Privileged Functions

- `pause`
- `unpause`
- `sweep`
- `updateMerkleRoot`

2.21.2 Issues & Recommendations

Issue #20	Rewards can be distributed twice if the contract is paused too late
Severity	LOW SEVERITY
Description	Unclaimed rewards are added to the next round. However, if the root is calculated before calling `pause`, some users could have claimed between the time the root was calculated and set to the Merkle tree. These users would then receive twice the amount of the previous round. This issue is only rated as low as `updateMerkleRoot` is only callable when the contract is paused, which shows that the right flow is the expected one. We have raised this issue it to ensure it is not forgotten.
Recommendation	Consider carefully pausing the contract before calculating the next root.
Resolution	ACKNOWLEDGED This is the expected flow.

Issue #21	Typographical error
Severity	INFORMATIONAL

Location	L34-35
	
	`//Handle when someone else accidentally transfers assets to this contract, or if we`
	`//need to migration to a new contract.`

Description	This should be changed to 'need to *migrate* to a new contract'.
Recommendation	Consider fixing the typographical error.
Resolution	ACKNOWLEDGED

2.22 Timelock

`Timelock` is a clean fork of Compound Finance's timelock. This is the most common contract used in DeFi to time lock governance access and is thus compatible with most third-party tools.

`Timelock` allows an administrator to set a delay before transactions are executed, which must be between 12 hours and 30 days. This prevents the administrator from executing transactions without first announcing them beforehand. Transactions can be queued by the administrator, and they will be executed after the delay has passed. If a transaction is not executed within the grace period, it is considered stale and will not be executed. This ensures that only transactions which have been properly announced and queued will be executed, preventing the administrator from executing unauthorized or malicious transactions.

The admin is the account which has been designated as the owner of the Timelock contract.

Parameter	Value	Description
Delay	12 hours	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	12 hours	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

2.22.1 Privileged Functions

- `setDelay [ timelock itself ]`
- `setPendingAdmin [ timelock itself ]`
- `acceptAdmin [ new owner ]`
- `queueTransaction [ owner ]`
- `cancelTransaction [ owner ]`
- `executeTransaction [ owner ]`

2.22.2 Issues & Recommendations

No issues found.

2.23 TimelockCallable

`TimelockCallable` is an abstract contract that is meant to be inherited by various contracts. It contains logic that allows certain functions to get only executed by the `Timelock`.

The timelock can be changed by the timelock by calling the setTimelock function.

2.23.1 Privileged Functions

- `setTimelock (onlyTimelock)`

2.23.2 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY