PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Wallchain (Paraswap Augustus Wrapper)

21 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 Wallchain's Paraswap Augustus wrapper 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.

1.1 Summary

Project Name	Wallchain (Paraswap Augustus wrapper)
URL	https://www.wallchain.xyz/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`SwapRouterManager`	`0xfaa746afc5ff7d5ef0aa469bb26ddd6cd8f13911`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	2	2	-	-
Low	1	1	-	-
Informational	6	4	-	2
Total	**12**	**10**	**-**	**2**

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 SwapRouterManager

ID	Severity	Summary	Status
01	HIGH	Excess `fromToken` might get stuck during `simpleBuy` and buy	✓ RESOLVED
02	HIGH	The contract does not work with `permit`	✓ RESOLVED
03	HIGH	The use of `tx.origin` is a risky approach	✓ RESOLVED
04	MEDIUM	Infinite approval is granted to the Paraswap Proxy	✓ RESOLVED
05	MEDIUM	Anyone can withdraw tokens transferred to the contract	✓ RESOLVED
06	LOW	The `receivedAmount` includes the fee	✓ RESOLVED
07	INFO	Tokens with a fee on transfer are not supported	ACKNOWLEDGED
08	INFO	`upgradeMaster` missing safeguard to `address(0)`	✓ RESOLVED
09	INFO	`withdrawEth` cannot be used by multi-signature contracts	ACKNOWLEDGED
10	INFO	Lack of events for `withdrawEth` and `withdrawAll`	✓ RESOLVED
11	INFO	Gas optimizations	✓ RESOLVED
12	INFO	Typographical errors	✓ RESOLVED

2 Findings

2.1 SwapRouterManager

`SwapRouterManager` is a wrapper implemented by the Wallchain team to integrate their cashback mechanism within the Paraswap protocol. Users can interact with the Paraswap August router through the Wallchain Router and earn cashback.

2.1.1 Privileged Functions

- `withdrawAll`
- `withdrawEth`
- `setShare`
- `setDexAgent`
- `upgradeMaster`
- `renounceOwnership`
- `transferOwnership`

2.1.2 Issues & Recommendations

Issue #01	**Excess `fromToken` might get stuck during `simpleBuy` and buy**
Severity	HIGH SEVERITY
Description	Unlike the other router calls, the `simpleBuy` and buy functions might leave a remaining amount of `fromToken` in the Paraswap router. This amount will then be sent back to `msg.sender`: `remainingAmount = Utils.tokenBalance(fromToken, address(this)); Utils.transferTokens(fromToken, msg.sender, remainingAmount);` However, the issue here is that the wrapper contract does not send it back to the initiator, which means that these funds will get stuck within the contract which amplifies Issue #05: *Anyone can withdraw tokens transferred to the contract*.
Recommendation	Consider transferring the leftover balance back to `msg.sender` after the swap was executed.
Resolution	RESOLVED The client has implemented a balance before/balance after approach.

Issue #02	The contract does not work with `permit`
Severity	HIGH SEVERITY
Description	`transferTokensIfNeeded` always calls `Utils.permit` with the `data.permit` parameter if one is given. The same `data.permit` parameter is then again forwarded to the router and used in `transferTokensFromProxy`. Due to the logic of how `permit` works, it is not possible to use the same permit data because the nonce will be mismatched in the 2nd call, essentially reverting the following: `require(signer == owner, "ERC20Permit: invalid signature");` https://github.com/soliditylabs/ERC20-Permit/blob/main/contracts/ERC20Permit.sol#L102
Recommendation	Consider removing the `permit` call within the `transferTokensIfNeeded` function.
Resolution	RESOLVED

Issue #03	The use of `tx.origin` is a risky approach
Severity	HIGH SEVERITY
Description	`SwapRouterManager` uses a well known Solidity variable called `tx.origin` which returns the initial EOA caller of the transaction. If a transaction is a chain transaction of multiple smart contracts, `tx.origin` will always return the initiator of the transaction, unlike `msg.sender` which returns the previous caller. Using `tx.origin` is very risky as we have seen before that certain authorizations can be bypassed or flows can be created that can lead to loss of funds. Within `SwapRouterManager`, `tx.origin` is used for example as a sender parameter when calling the `execute` function. If, for example, a user uses a multi-signature wallet or a smart contract wallet to initiate a transaction, `tx.origin` will be the user who signed the transaction within the multi-signature approval and not the multi-signature wallet, which means it will different from `msg.sender`. `tx.origin` is used as well within the `transferTokensIfNeeded` function.
Recommendation	Consider using `msg.sender` instead of `tx.origin`.
Resolution	RESOLVED

Issue #04	Infinite approval is granted to the Paraswap Proxy
Severity	MEDIUM SEVERITY
Description	`SwapRouterManager` must permit the Paraswap Proxy to transfer the users' funds in order to execute the action that was called to execute. In order to do this, `SwapRouterManager` must approve the Paraswap Proxy with a certain amount that is at least equal to the amount transferred by the user. Currently, the amount set for the approval is `MAX(uint256)` which is usually called an infinite approval as this number is very large. This approach is very risky because if the Paraswap Proxy contract gets compromised, it will be able to withdraw all the proxies that were approved by the `SwapRouterManager`. The approval is done at lines 123-125 within the `maybeApproveERC20` function. `if (token.allowance(address(this), tokenTransferProxy) < amount) { token.approve(tokenTransferProxy, UINT256_MAX); }`
Recommendation	Consider approving only the amount that is needed to execute the action within the `maybeApproveERC20` function. Additionally, consider resetting the approval before approving any new amounts. Paladin recommends using a library like SafeERC20 from Open Zeppelin to deal with the approvals.
Resolution	RESOLVED Only the amount that needs to be transferred is approved.

Issue #05	Anyone can withdraw tokens transferred to the contract
Severity	MEDIUM SEVERITY
Description	If a wallet transfers tokens to the contract by mistake, anyone will be able to withdraw them by carrying out a swap that has the token that was transferred by mistake as the target token. This behavior is possible due to the fact that the operations are using `balanceOf(address(this))` to transfer the result of the swap operation, and not the actual amount that was received as a result of the swap operation. To amplify this issue, a malicious sniping bot can scan the contract for any token balance and then simply input their own address as the beneficiary. The bot will thus receive the swapped amount directly while the contract will not receive any funds but still executes the transfer with the whole balance to `msg.sender`.
Recommendation	Consider using the actual received amount as the result of the swap operation instead of the balance of the contract. Moreover, it might make sense to limit `data.beneficiary` to the wrapper contract to ensure the contract will always receive back the desired tokens before transferring it out to the user.
Resolution	RESOLVED The client has implemented a balance before/balance after approach.

Issue #06	The `receivedAmount` includes the fee
Severity	LOW SEVERITY
Description	The Paraswap Router has two possibilities of taking the fee for a swap, from the `from` token or the `to` token. If the fee is taken from the `to` token, the user will receive the amount of tokens minus the Paraswap fees. The Paraswap Router does not update the `receivedAmount` variable within the swap operation, which will return an incorrect value that does not reflect what the user received. An example can be seen on this transaction: https://polygonscan.com/tx/0xc371f7397ab394668e10b9660d55ab1c4e8b1f189f363b673bc531036f6f9ba6 The user received 456531569702883196877O DAI but the `receivedAmount` variable is 45653157030859818647 85 as the fee of 6057149896015 is not subtracted in the return value. The issue is marked as Low severity because it does not affect the current `SwapRouterManager` implementation but it can become high if the return `receivedAmount` is used in the future within other contracts.
Recommendation	Consider using a balance before - balance after approach to determine the amount received.
Resolution	RESOLVED The client has implemented a balance before/balance after approach.

Issue #07	Tokens with a fee on transfer are not supported
Severity	INFORMATIONAL
Description	Tokens that have a fee on transfer are not supported by the `SwapRouterManager`. Additionally, the Paraswap protocol also does not support such tokens.
Recommendation	This issue is more informational for the reader of the Audit and for the Wallchain team.
Resolution	ACKNOWLEDGED

Issue #08	`upgradeMaster` missing safeguard to `address(0)`
Severity	INFORMATIONAL

Location	Line 111-113
	`address nextAddress = wchainMaster.nextAddress();`
	`if (address(wchainMaster) != nextAddress) {`
	`  wchainMaster = IWChainMaster(nextAddress);`

Description	`upgradeMaster` is used to upgrade the Wallchain Master in case of an upgrade. Currently, the upgrade does not check if the next address is different than `address(0)` — if the Wallchain Master returns `address(0)` when calling `nextAddress()`, the update will be successful but it will block the `SwapRouterManager`.
Recommendation	Consider if this is the right behavior, if not, consider adding a check within `upgradeMaster` to check if the `nextAddress` is different from `address(0)`.
Resolution	RESOLVED

Issue #09	`withdrawEth` cannot be used by multi-signature contracts
Severity	INFORMATIONAL
Description	`withdrawEth` uses the `.transfer` method to transfer the ETH balance out of the contract. This method forwards just 2300 gas to the receiver which can cause issues if the `msg.sender` is a multi-signature wallet or a smart contract.
Recommendation	Consider using `.call` instead of `.transfer` to withdraw the ETH.
Resolution	ACKNOWLEDGED The client has stated that this function is introduced for corner cases only.

Issue #10	Lack of events for `withdrawEth` and `withdrawAll`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. Additionally, it is not ideal to emit the same event with different parameters for different states. Each function should emit their own event.
Recommendation	Consider adding events for the above functions, change `EventMessage` and associate a unique event for each function.
Resolution	RESOLVED

Issue #11	Gas optimizations
Severity	INFORMATIONAL
Description	We have consolidated the sections which can be further optimized for gas usage below. The `tokenTransferProxy` and `router` variables can be `immutable`. Initializing a variable with a default value is obsolete. E.g. `for (uint256 i = 0; i < tokens.length; i++) { the uint256 i =0` is obsolete as i is already 0, therefore this adds unnecessary gas consumption.
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	RESOLVED

Issue #12	Typographical errors
Severity	INFORMATIONAL
Description	Various functions can be made external. - `withdrawEth` - `simpleSwap` - `buy` - `multiSwap` - `megaSwap`
Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY