PALADIN
BLOCKCHAIN SECURITY

# Smart Contract Security Assessment

Final Report

## For ApeSwap (WallChain Integration)

14 May 2022

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 ApeSwap 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** | ApeSwap |
| **URL** | https://apeswap.finance/ |
| **Platform** | Polygon |
| **Language** | Solidity |

## 1.2 Contracts Assessed

| Name | Contract | Live Code Match |
|---|---|---|
| `RouterManager` | `0x5471f99bcb8f682f4fd2b463fd3609dadd56a929` | MATCH |
| `WChainMaster` | `0x178C075a1d413f8637B3C6623c9501167D457bf3` | MATCH |

## 1.3 Findings Summary

| Severity | Found | Resolved | Partially Resolved | Acknowledged (no change made) |
|---|---|---|---|---|
| High | 0 | - | - | - |
| Medium | 2 | 2 | - | - |
| Low | 1 | 1 | - | - |
| Informational | 12 | 11 | 1 | - |
| **Total** | **15** | **14** | **1** | **-** |

### 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 RouterManager

| ID | Severity | Summary | Status |
|---|---|---|---|
| 01 | INFO | Typographical errors | RESOLVED |
| 02 | INFO | `swapExactTokensForTokens` can be made `external` | RESOLVED |
| 03 | INFO | Gas optimizations | RESOLVED |
| 04 | INFO | `maybeApproveERC20` does not reset approval if there is a small remainder | RESOLVED |
| 05 | INFO | Lack of events for upgradeMaster and setDexAgent | RESOLVED |

### 1.3.2 WChainMaster

| ID | Severity | Summary | Status |
|---|---|---|---|
| 06 | MEDIUM | The user can steal the complete share for WChain | RESOLVED |
| 07 | MEDIUM | The flash loan does not explicitly support the fee rate of all routers | RESOLVED |
| 08 | LOW | `userShare`, `treasuryShare` and `FACTORY` are `private` | RESOLVED |
| 09 | INFO | Lack of events for `setShares`, `pushUpgrade`, `withdraw` and `withdrawAll` | RESOLVED |
| 10 | INFO | Lack of validation | RESOLVED |
| 11 | INFO | Typographical errors | RESOLVED |
| 12 | INFO | Lack of `safeTransfer` usage within withdrawals and various other locations | RESOLVED |
| 13 | INFO | `FACTORY` can be made immutable | RESOLVED |
| 14 | INFO | Gas optimizations | PARTIAL |
| 15 | INFO | `execute` does a useless `maxLen` calculation | RESOLVED |

# 2 Findings

## 2.1 RouterManager

The `RouterManager`, also called the Wallchain Router Proxy, redirects all swaps to the ApeSwap DEX router and triggers the Wallchain Master to capture back running profits after any swap.

Backrunning profit-taking is a strategy that does not affect the user in any way except for increased gas cost as it's executed after the user swap. The most common back running strategy is arbitraging between two pairs once one of the pairs has been put out of balance by the user their swap.

From a high level, the Wallchain API calculates various back-running strategies off-chain before any swap. If a profitable strategy is found, it proposes to ApeSwap to execute the swap over the `RouterManager` compared to the normal router. With this execution, the `RouterManager` receives a payload which encodes how to execute the back-running strategy: Which swaps to make, which amounts to swap and whether a flash loan might be required.

The contract does not support tokens with a fee on transfer.

### 2.1.1 Privileges

The following functions can be called by the owner of the contract:

- `transferOwnership`
- `renounceOwnership`
- `upgradeMaster`
- `setDexAgent`
- `addBackupRouter`
- `removeBackupRouter`

### 2.1.2 Issues & Recommendations

| Issue #01 | Typographical errors |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | The contract contains a number of typographical errors which we have consolidated below in a single issue in an effort to keep the report size reasonable.<br><br>Line 12<br>`uint256 constant UINT256_MAX = 2**256 - 1;`<br><br>This can be simplified to `type(uint256).max`.<br><br>Line 19<br>`address dexRouterAddress,`<br><br>This variable can be provided as the `IUniswapV2Router02` type to avoid casting it later on.<br><br>Line 21<br>`address wchainMasterAddress`<br><br>This variable can be provided as the `IWChainMaster` type to avoid casting it later on.<br><br>Line 28<br>`receive() external payable {}`<br><br>There is no value in having a `receipt` function on this contract. It might cause users to accidentally send tokens to it.<br><br>Line 41<br>`function maybeApproveERC20(address token) private {`<br><br>`token` can be directly cast to `IERC20`. |
| **Recommendation** | Consider fixing the typographical errors. |
| **Resolution** | RESOLVED |

| Issue #02 | **`swapExactTokensForTokens` can be made `external`** |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | Functions that are not used within the contract but only externally can be marked as such with the `external` keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases. |
| **Recommendation** | Consider marking the functions mentioned above as `external`. |
| **Resolution** | RESOLVED |

| Issue #03 | Gas optimizations |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | The contract contains multiple sections of code that could be further optimized for gas efficiency. We have consolidated these in a single issue in an effort to keep the report brief and readable.<br><br>The most generic gas optimization is to redesign the codebase into having a single router. There is no reason why this `RouterManager` must hook into the existing router. A Uniswap deployment can in fact have many routers. In this case, it would be significantly more gas efficient if the `coverUp` modifier is simply added to a copy of the existing router. By moving the hooks into the main router, tokens with a fee on transfer could also be supported which we would consider a huge benefit as they tend to have frequent arbitrage opportunities.<br><br>Lines 30, 51, 54<br>`modifier coverUp(bytes memory masterInput) {`<br>`address[] memory path,`<br>`bytes memory masterInput`<br><br>Various parameters in the contract can be marked as `calldata`. `swapExactTokensForTokens` needs to be made external for this to be possible.<br><br>Line 56<br>`amounts = dexRouter.getAmountsOut(amountIn, path);`<br><br>These amounts are also returned by the underlying router. It is not necessary to fetch them twice. The following requirement on line 57 is also redundant.<br><br>Line 131<br>`path[path.length - 1] == dexRouter.WETH(),`<br><br>WETH can be cached as an immutable variable throughout the contract to save a significant amount of gas. |

Line 167

```
amounts = dexRouter.getAmountsOut(amountIn, path);
```

These amounts are also returned by the underlying router. It is not necessary to fetch them twice. The requirements on line 163 and line 168 are also redundant.

Line 196

```
amounts = dexRouter.getAmountsIn(amountOut, path);
```

These amounts are also returned by the underlying router. It is not necessary to fetch them twice. The requirements on line 195 and line 197 are also redundant.

| | |
|---|---|
| **Recommendation** | Consider implementing the gas optimizations mentioned above. |
| **Resolution** | RESOLVED<br>Many of these gas optimizations have been implemented. The codebase was not merged with the main router in line with keeping the system as modular as possible. |

| Issue #04 | **`maybeApproveERC20` does not reset approval if there is a small remainder** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Location** | Lines 43-45<br>`if (IERC20(token).allowance(address(this), address(dexRouter)) == 0) {`<br>`    IERC20(token).approve(address(dexRouter), UINT256_MAX);`<br>`}` |
| **Description** | Instead of always approving the router, approval is only granted if the current allowance is zero. If the current allowance is however abruptly small, smaller than the amount that is supposed to be sent to the router, this will cause the re-approval to not be triggered.<br><br>It should be noted that this is a purely theoretical issue as we expect an approval to never come down anywhere close to zero. |
| **Recommendation** | Consider passing an amount parameter to `maybeApproveERC20` and increase the approval whenever the allowance is smaller than the amount. Personally we would likely simply have a `mapping(address => bool)` to keep track of which tokens have received approval already.<br><br>In general, the whole approval flow can be circumvented by adding the `coverUp` modifier to the existing router (or deploying a second full router with the `coverUp` modifier). |
| **Resolution** | RESOLVED<br><br>There is now a check if the allowance still covers the input amount instead of zero. |

| Issue #05 | Lack of events for `upgradeMaster` and `setDexAgent` |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | Functions that affect the status of sensitive variables should emit events as notifications. |
| **Recommendation** | Add events for the functions. |
| **Resolution** | RESOLVED |

## 2.2 WChainMaster

`WChainMaster` is the contract that does the grunt work of executing the back-running strategy. After each `RouterManager` call, the `execute` function is called with the off-chain calculated back-running payload.

Each back-running payload contains a set of proposed trades. During the payload execution, `WChainMaster` iterates over all of them and picks the single most profitable trade at the time of execution.

Each backrunning strategy has a simple objective: increase the number of "input tokens" required for the strategy. Such input tokens are usually flash loaned in the first step of the strategy.

After such a strategy has been successfully executed, the contract ends up with more tokens than it initially had. These tokens are allocated to three different recipients:

- The user: 40%
- ApeSwap: 40%
- WallChain: 20%

These distributions can be changed by WallChain.

All through the `WChainMaster` is not crazily optimized for maximum value extraction (no multi-routing, no support of non-Uniswap routes, no flash loaning from cheaper/free sources), it is engineered to slowly build its own financing and depend less and less upon flash loans.

By default, the profit share that is meant for WallChain is kept in the `WChainMaster` and can be used on the next arbitrage opportunity. WallChain is therefore effectively earning 0.3% interest every time the liquidity within the `WChainMaster` is used by them. This is an admirable optimization.

### 2.2.1 Privileges

The following functions can be called by the owner of the contract:

- `transferOwnership`
- `renounceOwnership`
- `setShares`
- `pushUpgrade`
- `withdraw`
- `withdrawAll`

### 2.2.2 Issues & Recommendations

| Issue #06 | The user can steal the complete share for WChain |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | Presently the `shareProfit` function swaps the profit over a route generated by the API. A malicious router could route this profit to a pair and destination token only they possess. By doing this they can also capture the DEX profit share. |
| **Recommendation** | This issue is quite fundamental, at the end of the day the `WChainMaster` is not omnipotent and the user could just deploy their own instance to capture 100% of this profit. We consider it sufficient if the client takes this restriction and lack of enforceability into consideration. |
| **Resolution** | RESOLVED<br>The client has indicated that this vector is not within their threat model as a user can always decide to avoid using their router. We agree, especially in light that there is no perfect way to resolve this issue. |

| Issue #07 | The flash loan does not explicitly support the fee rate of all routers |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Location** | Line 390<br>`uint256 amountRequired = ((amountToken * 10000) / 9975) + 1;` |
| **Description** | The flash loan does not explicitly support the fee rate of all routers. If a DEX is ever flashloaned from with a fee-rate smaller than 0.25%, the clients would have to still send them a 0.25% fee.<br><br>It also does not= support any Uniswap deployment with a fee greater than 0.25%. |
| **Recommendation** | Consider whether this would ever pose an issue. If so, consider potentially keeping track of a mapping of factories and their respective fee. |
| **Resolution** | RESOLVED<br><br>The client can now adjust the fee over time. |

| Issue #08 | `userShare, treasuryShare` and `FACTORY` are `private` |
|---|---|
| **Severity** | LOW SEVERITY |
| **Description** | Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts. |
| **Recommendation** | Consider marking the variables as public. |
| **Resolution** | RESOLVED<br><br>The `userShare` and `treasuryShare` are now provided off-chain while the factory parameter has been made public. |

| Issue #09 | **Lack of events for `setShares`, `pushUpgrade`, `withdraw` and `withdrawAll`** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Functions that affect the status of sensitive variables should emit events as notifications. |
| **Recommendation** | Add events for the functions. |
| **Resolution** | RESOLVED |

| Issue #10 | **Lack of validation** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | The contract contains functions with parameters which are not properly validated. Having unvalidated parameters could allow the governance or users to provide variable values which are unexpected and incorrect. This could cause side-effects or worse exploits in other parts of the codebase.<br><br>Line 76<br>`function setShares(uint256 newUserShare, uint256 newTreasuryShare)`<br><br>The `newUserShare` and `newTreasuryShare` should never sum to a total greater than 100. |
| **Recommendation** | Consider validating the function parameters mentioned above. |
| **Resolution** | RESOLVED<br>The sum of the two has now been made constant to 80. The router provides the division between the two variables and does validation. This would be insufficient by itself as the user can avoid providing these parameters from the router by calling the `WChainMaster` directly, but the `WChainMaster` also does a final check on them, which strictly enforces this requirement. |

| Issue #11 | Typographical errors |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | The contract contains a number of typographical errors which we have consolidated below in a single issue in an effort to keep the report size reasonable.<br><br>Line 3<br>`pragma experimental ABIEncoderV2;`<br><br>We are unsure if this pragma is necessary on this codebase.<br><br>Line 51<br>`// Address of the next Versoin WChainMaster`<br><br>This line should say version, not Versoin.<br><br>Lines 60-62<br>`address private FACTORY;`<br>`constructor(address _factory) Ownable() {`<br><br>The `FACTORY` and `_factory` variables can be set as the `IUniswapV2Factory` type to reduce casting and simplify the codebase.<br><br>Lines 66-74<br><br>Various variables can be set as their explicit types to reduce the frequency of needing to cast them later on. These types include `IWChainPair[]`, `IWChainRouter[]` and `IWChainERC20`. |

Line 96

```
address[] calldata tokenAddresses,
```

Line 105

```
function withdrawAll(address[] calldata tokenAddresses)
external onlyOwner
```

The `tokenAddresses` parameter can be provided as the `IWChainERC20` type to avoid casting it later on in both of these locations.

Line 143

```
(reserveIn, reserveOut, ) = pair.getReserves();
```

These variables in fact represent the exact opposite of their names in this else branch.

Line 230

```
address flashPair = abi.decode(trade.extraParams[0],
(address));
```

`trade.extraParams[0]` can be directly decoded as a `IWChainPair`.

Line 238

```
IWChainPair(flashPair).swap(
```

The `flashPair` has already been cast to pair on line 232. Consider simply using `pair`.

**Recommendation** Consider fixing the typographical errors.

**Resolution** RESOLVED

Most of the errors have been corrected but we have resolved these issues as they are only of informational severity.

| Issue #12 | **Lack of `safeTransfer` usage within withdrawals and various other locations** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Location** | Example: Line 101<br>`erc20.transfer(msg.sender, amounts[i]);`<br><br>Example: Line 105<br>`erc20.transfer(msg.sender, erc20.balanceOf(address(this)));`<br><br>Example: Line 203<br>`IWChainERC20(bestTrade.firstTokenAddress).transfer(` |
| **Description** | In the `withdraw` and `withdrawAll` functions, the `transfer` method is used to transfer tokens from the contract to the owner. This will not work for tokens that return `false` on transfer (or malformed tokens that do not have a return value). |
| **Recommendation** | Consider using `safeTransfer` instead of `transfer`. There are more locations than just the examples provided above. The clients should consider fixing all locations. |
| **Resolution** | RESOLVED |

| Issue #13 | **FACTORY can be made immutable** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Variables that are only set in the constructor but never modified can be indicated as such with the immutable 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 immutable. |
| **Resolution** | RESOLVED<br>The factory can now be changed therefore this issue is no longer present. |

| Issue #14 | Gas optimizations |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | The contract contains multiple sections of code that could be further optimized for gas efficiency. We have consolidated these in a single issue in an effort to keep the report brief and readable.<br><br>Line 182<br>`function shareProfit(`<br><br>It might not make sense to perform a swap for every recipient that receives profit. Instead, one could consider performing the swap to `address(this)` and then transferring out the shares. Users should be very careful about tokens with a fee on transfer and malicious trades however.<br><br>Line 149<br>`return (pair.token0(), currentAmount, 0);`<br><br>`token0()` can be cached as it is used earlier in the function. |
| **Recommendation** | Consider implementing the gas optimizations mentioned above. |
| **Resolution** | PARTIALLY RESOLVED<br><br>The second issue has been resolved. |

| Issue #15 | `execute` does a useless `maxLen` calculation |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | The `execute` function calculates the maximum path length. However, this `maxLen` variable is not used afterwards. |
| **Recommendation** | Consider removing the calculation and variable if it is not used. |
| **Resolution** | RESOLVED<br>This calculation has been removed. |


PALADIN
BLOCKCHAIN SECURITY