PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For SwapperChan

20 November 2021

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.

1 Overview

This report has been prepared for SwapperChan on the Boba 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	SwapperChan
URL	https://swapperchan.com/
Platform	Boba Network
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
SwapperChanRouter	0x215EC743F2316A0fa4a6Af5A7C331E859f4F5E2b	MATCH
ERC20	Dependency	MATCH
SwapperChanPair	Dependency	MATCH
SwapperChanFactory	0x3d97964506800d433fb5DbEBDd0c202EC9B62557	MATCH
Library	Dependency	MATCH
Math, SafeMath, TransferHelper, UQ112x112	Dependency	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	1	1	-	-
Low	2	-	-	2
Informational	9	2	-	7
Total	**12**	**3**	**-**	**9**

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 with 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 SwapperChanRouter

ID	Severity	Summary	Status
01	MEDIUM	Router fee calculation is wrong for swaps with exact output	RESOLVED
02	LOW	Router swap fee can be bypassed by directly interacting with LP contract	ACKNOWLEDGED
03	LOW	Swaps will fail if the user does not set the minimum slippage to include the router fee percentage	ACKNOWLEDGED
04	INFO	Phishing is possible by a malicious frontend by adjusting routes, tokens or `from` parameters (also present in Uniswap)	ACKNOWLEDGED
05	INFO	Unnecessary fee calculation even if fee percentage is zero	ACKNOWLEDGED
06	INFO	_WETH that could be used in the constructor upon contract deployment could possibly be an unverified token	ACKNOWLEDGED
07	INFO	Constructor inputs lack a zero address input verification	RESOLVED
08	INFO	Some router functions are not being called internally and can be declared `external`	ACKNOWLEDGED
09	INFO	Function `addLiquidity` still does not support reflective tokens upon transfer	ACKNOWLEDGED
10	INFO	Lack of event emissions for sensitive changes	RESOLVED

1.3.2 ERC20

ID	Severity	Summary	Status
11	INFO	`permit` can be frontrun to prevent someone from calling `removeLiquidityWithPermit` (also present in Uniswap)	ACKNOWLEDGED

1.3.3 Pair

ID	Severity	Summary	Status
12	INFO	Pairs without supply but with a partial reserve might crash the frontend if the user wants to swap on this pair (present in most frontends)	ACKNOWLEDGED

1.3.4 SwapperChanFactory

No issues found.

1.3.5 Library

No issues found.

1.3.6 Math, SafeMath, TransferHelper, UQ112x112

No issues found.

2 Findings

2.1 SwapperChanRouter

The SwapperChan AMM protocol, forked with some changes from Uniswap, uses the SwapperChanRouter as an entry point for users to exchange tokens. The SwapperChanRouter is responsible for determining the swap rate and allowing for user-interactions to be done with safety checks. More specifically, the SwapperChanRouter allows routers to add liquidity, remove liquidity and swap tokens.

One change introduced to the router is an additional swap fee which is deducted from the amount provided by the user. For swaps that use an exact `amountIn`, the actual `amountIn` used for the actual swap is the `amountIn` specified to the function subtracted by the fee. For swaps that use an exact `amountOut`, the actual `amountIn` is the sum of the input token returned by `getAmountsIn`, and the fee. The fee is initialized at 0.1%, but can be modified up to 0.5%.

2.1.1 Issues & Recommendations

Issue #01	Router fee calculation is wrong for swaps with exact output
Severity	MEDIUM SEVERITY

Description	The calculation for the fee for swaps with an exact output causes higher fees than compared to the fee for swaps with an exact input, given the same input amount.
	
	`Exact input: Line 239`
	`uint fees = amountIn.mul(swapFee).div(1000);`
	
	`Exact output: Line 265~`
	`uint amountIn = amounts[0].mul(1000).div(1000 - swapFee);`
	`uint fees = amountIn.mul(swapFee).div(1000);`
	
	Example:
	
	Assume that the swapFee is set to 5 (0.5%).
	
	For an `amountIn` of value of 1000, the fee for exact input is 5 of the input token.
	
	For an `amounts[0]` value of 1000, the fee for exact output is 5.0251 of the input token.
	
	There is a discrepancy between 0.5% and 0.50251% fees between the 2 types of swaps.

Recommendation	For exact output swaps, the fee has to be calculated the same way as the fee of exact input swaps. `uint fees = amounts[0].mul(swapFee).div(1000);`
Resolution	RESOLVED The recommended fee calculation has been implemented.

Issue #02	Router swap fee can be bypassed by directly interacting with LP contract
Severity	LOW SEVERITY
Description	If a user would like to swap tokens in a specific LP, they can bypass the additional swap fee imposed in the router by transferring the in tokens to the LP contract, followed by calling the swap function.
Recommendation	If this additional fee is to be enforced on all swaps, it has to be enforced at the LP contract level.
Resolution	ACKNOWLEDGED

Issue #03	Swaps will fail if the user does not set the minimum slippage to include the router fee percentage
Severity	LOW SEVERITY
Description	Unless the frontend UI accounts for the slippage to be the user defined slippage + the router fee percentage, swaps can fail due to too high `amountOutMin` or too low `amountInMax`. For example, for a `swapExactTokensForTokens`, if the router fee is set to 0.5%, if the user sets the slippage to 0.5% on the frontend, which is used for the calculation of `amountOutMin`. If the swap's actual slippage is 0.4%, the swap will still fail as the actual slippage is 0.9%.
Recommendation	The frontend should inform the user about the router fee, and that fee percentage should be included as part of the slippage used for the calculation of `amountOutMin` or `amountInMax`.
Resolution	ACKNOWLEDGED

Issue #04	**Phishing is possible by a malicious frontend by adjusting routes, tokens or `from` parameters (also present in Uniswap)**
Severity	INFORMATIONAL
Description	A malicious (e.g. compromised) frontend can easily mislead users in approving malicious transactions, even if the router matches the address described in this report. An obvious example of how this can be done is by changing the `to` parameter which indicates to whom tokens or liquidity has to be sent. Other ways to phish could include using malicious routes or tokens.
Recommendation	Consider carefully protecting the frontend and ideally having an unchangeable IPFS fallback implementation for it. Users should also verify that they are on the correct website when doing a swap.
Resolution	ACKNOWLEDGED

Issue #05	**Unnecessary fee calculation even if fee percentage is zero**
Severity	INFORMATIONAL
Location	`Line 238 (example)` `if (feeAddress != address(0) && swapFee != 0) {`
Description	Currently, the fee calculation is skipped if the fee address is zero, but still calculated even if the fee percentage is zero.
Recommendation	Add an extra condition in the `if` statement to only calculate and use the fee if either the fee address and fee percentage is not zero.
Resolution	ACKNOWLEDGED

Issue #06	_WETH that could be used in the constructor upon contract deployment could possibly be an unverified token
Severity	INFORMATIONAL
Description	If the contract will be deployed, a constructor will require an input of _WETH address for the network's native token for use in the router's swapping functions.
Recommendation	An unverified token could potential contain anything and should be avoided. Should the network native token be unverified, attempt to decompile the ByteCode to see if there is anything malicious.
Resolution	ACKNOWLEDGED

Issue #07	Constructor inputs lack a zero address input verification
Severity	INFORMATIONAL
Description	Inadvertently inputting zero address in the constructor will render the contract unusable and will require redeployment.
Recommendation	Consider implementing: `require( _factory != address(0), "Can not be zero address" ), and a require( _WETH != address(0), "Can not be zero address" ) statement.`
Resolution	RESOLVED Non-zero address checks have been added in the constructor.

Issue #08	**Some router functions are not being called internally and can be declared `external`**
Severity	INFORMATIONAL
Description	The following `public` functions are never called by the contract and thus should be declared `external` to save gas: • `quote` • `getAmountOut` • `getAmountIn` • `getAmountsOut` • `getAmountsIn`
Recommendation	Use the `external` attribute for functions never called from the contract.
Resolution	ACKNOWLEDGED

Issue #09	**Function `addLiquidity` still does not support reflective tokens upon transfer**
Severity	INFORMATIONAL
Description	The current `addLiquidity` function always assumes tokens are not reflective, thus resulting in waste should users be opting to transact with reflective tokens.
Recommendation	The client may opt to include the following function that could help calculate the quotes beforehand for transfer tax tokens. `addLiquiditySupportingFeeOnTransfer` first sends the `feeToken` to the pair, then it scales down the amount of `tokenB` to sent to the pair based on how much arrived. You can read more about this issue here: https://github.com/Uniswap/v2-periphery/issues/106

Interface:

```
function addLiquiditySupportingFeeOnTransfer(
    address feeToken,
    address tokenB,
    uint amountFeeTokenDesired,
    uint amountBDesired,
    uint amountFeeTokenMin,
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountFeeToken, uint amountB, uint liquidity);
```

Router:

```
function addLiquiditySupportingFeeOnTransfer(
    address feeToken,
    address tokenB,
    uint amountFeeTokenDesired,
    uint amountBDesired,
    uint amountFeeTokenMin,
    uint amountBMin,
    address to,
    uint deadline
) external virtual override ensure(deadline) returns (uint amountFeeToken,
uint amountB, uint liquidity) {
    (amountFeeToken, amountB) = _addLiquidity(feeToken, tokenB,
amountFeeTokenDesired, amountBDesired, amountFeeTokenMin, amountBMin);
    address pair = UniswapV2Library.pairFor(factory, feeToken, tokenB);
    uint256 before = IERC20(feeToken).balanceOf(pair);
    TransferHelper.safeTransferFrom(feeToken, msg.sender, pair,
amountFeeToken);
    uint amountBReduced =
IERC20(feeToken).balanceOf(pair).sub(before).mul(amountB).div(amountFeeToken)
; // Pro-rata adjustment
    if(amountBReduced < amountB) // We only want to decrease amount B
        amountB = amountBReduced;
    TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
    liquidity = IUniswapV2Pair(pair).mint(to);
}
```

Resolution ACKNOWLEDGED

Issue #10	Lack of event emissions for sensitive changes
Severity	INFORMATIONAL
Description	There is a lack of event emission for the following functions that result in sensitive changes and the behavior of the contract. • `setFeeAddress` • `setSwapFees`
Recommendation	Emit events with the changed values.
Resolution	RESOLVED Events are now emitted.

2.2 ERC20

The SwapperChanERC20 is an implementation of the ERC-20 Token Standard. It is a clean copy of the related Uniswap contract.

2.2.1 Issues & Recommendations

Issue #11	**Permit can be frontrun to prevent someone from calling `removeLiquidityWithPermit` (also present in Uniswap)**
Severity	INFORMATIONAL

Location	Line 82~
	`function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external {`
	`    require(deadline >= block.timestamp, 'SwapperChan: EXPIRED');`
	`    bytes32 digest = keccak256(`
	`            abi.encodePacked(`
	`            '\x19\x01',`
	`            DOMAIN_SEPARATOR,`
	`            keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadline))`
	`            )`
	`    );`
	`    address recoveredAddress = ecrecover(digest, v, r, s);`
	`    require(recoveredAddress != address(0) && recoveredAddress == owner, 'SwapperChan: INVALID_SIGNATURE');`
	`    _approve(owner, spender, value);`
	`}`

Description	Currently if `permit` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `permit` transactions in the mempool and execute them before a contract can. The implications of this issue is that a bad actor could prevent a user from removing liquidity with a `permit` through the router. It is a denial of service attack which is present in all AMMs but which we have yet to witness being used since there is no profit from it.
Recommendation	Consider this issue if there are ever complaints by users that their `removeLiquidityWithPermit` transactions are failing. It could be the case that someone is using this vector against them. We do not recommend changing this behavior since it would cause a lot of extra work modifying the frontend to account for new permit behavior. This issue is also present in Uniswap after all.
Resolution	ACKNOWLEDGED

2.3 Pair

The SwapperChan Pair is the core component of the SwapperChan AMM protocol, it represents a pair of two tokens. People can add liquidity in this pair by depositing both tokens in equally valued proportion for others to swap against. It is a clean fork from Uniswap.

Fees are 0.2% of each swap. 1/6 of fees go to the fee address, whilst LP holders receive 5/6 of fees. This fee is different from the fee at the Router contract.

2.3.1 Issues & Recommendations

Issue #12	**Pairs without supply but with a partial reserve might crash the frontend if the user wants to swap on this pair (present in most frontends)**
Severity	INFORMATIONAL
Description	A malicious DoS attack we've witnessed in practice is when a project wants to go live through a presale, people can instantiate the pair while there are no tokens yet. The malicious party will then sent some of the counterparty token to this pair so it has a partial balance (eg. 0.1 ETH and 0 tokens). When `sync()` is then called, the pairs' reserves are updated to account for this balance. Due to a division by zero exception, many frontends can not properly account for this state and will go through a blank page, preventing the original project from adding liquidity through the frontend.
Recommendation	Consider checking whether this is present in the frontend and adding a division by zero handler.
Resolution	ACKNOWLEDGED

2.4 SwapperChanFactory

The SwapperChanFactory is the management contract for the SwapperChan AMM. It keeps track of all SwapperChan Pairs and allows users to create new ones. Any SwapperChan created through the verified factory is immediately verified as well, since the pair is deployed by the verified factory.

2.4.1 Issues & Recommendations

No issues found.

2.5 Library

The SwapperChanLibrary contract is a dependency contract used to calculate the appropriate trading rates. It is used by the SwapperChanRouter to calculate how many tokens should be sent to the pairs and is thus an important component of the user-facing aspect of the system. The parts of the code that account for the calculation with fee have been modified to account for the change of 0.3% to 0.2% swap fees.

2.5.1 Issues & Recommendations

No issues found.

2.6 Math, SafeMath, TransferHelper, UQ112x112

Math, SafeMath, TransferHelper and UQ112x112 are various helper libraries which are each identical to the Uniswap implementation.

2.6.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY