XSEC FINANCE

Smart Contract Audit

Date	June 2021

GROWTHDEFI WHEAT CONTRACT AUDIT

This report contains the results of our collaboration with GrowthDeFi in order to review the WHEAT protocol.

1 | EXECUTIVE SUMMARY

This report contains the results of our collaboration with GrowthDeFi in order to review the WHEAT protocol.

Sergii Kravchenko and David Oz Kashi conducted the review over **three weeks**. The review was completed by **Dominik Muhs, Dominik Muhs, and Sergii Kravchenko**. The review lasted 30 person-days.

We spent the first week getting to know the GrowthDeFi system and the economic incentives it offers.

The second week saw us inspect the strategy token contracts and their interactions with third party infrastructure. We also examined their impact on the rest of our system. We also examined the distribution of rewards as well as potentially vulnerable user flows.

The third week was devoted to the collection of fees and we inspected the peripheral infrastructure a little more.

2 | SCOPE

Our review focused on the audit branch, specifically, commit hash **8360ac0a537589bb974e8a5a169bb3e7c95d2857.**

You can find the Appendix with a list of files within scope.

2.1 | Objectives

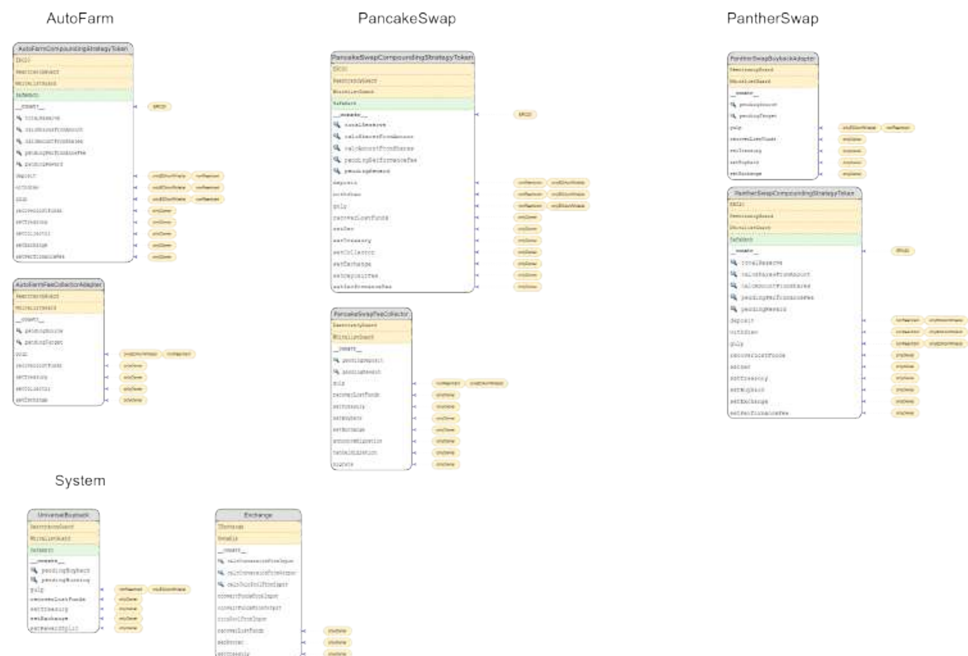
We identified these priorities together with the GrowthDeFi Team for our review:

Make sure that the system is consistent with its intended functionality and does not have any unintended side effects.

Identify vulnerabilities specific to smart contract systems as described in Smart Contract Best Practices and Smart Contract Weakness Classification Registry.

The prevention of attacks that could lead to the loss or corruption of user funds and/or system resources has been a key focus.

3 | SYSTEM OVERVIEW



4 | SECURITY SPECIFICATION

This section describes the security implications of the system being audited. This section is not intended to replace documentation. This section identifies security properties that have been validated by the audit team.

4.1 | Actors

Below are the relevant actors and their abilities:

- User/EOA
- Peripheral Project
- Strategy Token
- Fee Collector
- Buyback Actor

4.1 | Trust Assumptions

Security of the system heavily depends on security of the third parties. It is assumed that tokens and exchanges will exhibit consistent and correct behavior, e.g. In terms of token balances, transfers and transactions. We strongly recommend that tokens are thoroughly vetted before they are integrated into the GrowthDeFi ecosystem. The checklist could include the following:

- Examining audit reports prepared by independent, well-respected professionals.
- Examining the adherence of standard interfaces (ERC-20/BEP-20 MasterChef, Uniswap token couple),
- Ensure that security-critical components cannot be upgraded
- Assist the candidate's team in developing a plan to deal with security incidents.
- It is important to ensure that there is a contact person who can handle legal issues.

Trust is also placed on the functionality and availability of tx.origin. This may change in the future. The issues section provides more information.

5 | RECOMMENDATIONS

5.1 | Test suite improvements

Description

This stage does not include all of the test suites. Complex systems like GrowthDefi that interact with multiple DeFi protocols and use many modules, require a comprehensive test suite. This can be verified using solidity-coverage tool. It includes both unit and integration tests. This will ensure the correctness and safety of the core logic as well as any interface issues. A complete test suite can help to prevent problems that may be difficult to spot with manual reviews, as we have seen in smart contract incidents.

5.1 | Test suite improvements

Description

There are many code areas that are very similar across the code base, particularly in the strategy token contract code. The possibility of duplicate code in the future could cause problems as it is possible to fix bugs or make system behaviour inconsistent.

Examples.In the strategy token contracts:

- `calcSharesFromAmount` and other view functions
- Control `flow` and `deposit` checks
- `Gulp` conversions between reward- and reserve-tokens
- Business logic for `recoverLostFunds`

Recommendation

It is recommended that you deduplicate your codebase by inheriting generic contract contracts that provide common control flow, business logic actions, and other functionality. Then, override local methods to implement the specific strategy token behavior for each third-party interaction.

This will decrease the likelihood of future bugs, and increase maintainability and extensibility of the codebase.

6 | FINDINGS

Each issue is assigned a severity:

- **Minor** problems are subjective. These are usually suggestions about best practices or readability. These issues should be addressed by code maintainers.
- **Medium** issues are objective, but they are not security vulnerabilities. These issues should be addressed, unless there are compelling reasons not to.
- Security vulnerabilities are critical issues that can't be exploited directly or require special conditions to be exploited. All of these **Major** problems should be addressed.
- Security vulnerabilities that could be exploited to cause **Critical** issues need to be addressed.

6.1 | Frontrunning attacks by the `Owner` Major

Description

The owner has few options for attack vectors:

1 | All strategies come with fees and rewards. PancakeSwap also has deposit fees. The default deposit fees are zero and the maximum is 5%

wheat-v1-core-audit/contracts/PancakeSwapCompoundingStrategyToken.sol:L29-L33.

```
uint256 constant MAXIMUM_DEPOSIT_FEE = 5e16; // 5%
uint256 constant DEFAULT_DEPOSIT_FEE = 0e16; // 0%

uint256 constant MAXIMUM_PERFORMANCE_FEE = 50e16; // 50%
uint256 constant DEFAULT_PERFORMANCE_FEE = 10e16; // 10%
```

If a user deposits tokens expecting to pay no deposit fees, the owner may frontrun the deposit to increase fees up to 5%. This fee can be significant if the deposit is large enough.

2 | The reward tokens can be exchanged for reserve tokens in the gulp function:

wheat-v1-core-audit/contracts/PancakeSwapCompoundingStrategyToken.sol:L218-L244

```
function gulp(uint256 _minRewardAmount) external onlyEOAorWhitelist nonReentrant
{
  uint256 _pendingReward = _getPendingReward();
  if (_pendingReward > 0) {
      _withdraw(0);
  }
  {
      uint256 _totalReward = Transfers._getBalance(rewardToken);
      uint256 _feeReward = _totalReward.mul(performanceFee) / 1e18;
      Transfers._pushFunds(rewardToken, collector, _feeReward);
  }
  if (rewardToken != routingToken) {
      require(exchange != address(0), "exchange not set");
      uint256 _totalReward = Transfers._getBalance(rewardToken);
      Transfers._approveFunds(rewardToken, exchange, _totalReward);
      IExchange(exchange).convertFundsFromInput(rewardToken, routingToken, _totalReward, 1);
  }
  if (routingToken != reserveToken) {
      require(exchange != address(0), "exchange not set");
      uint256 _totalRouting = Transfers._getBalance(routingToken);
      Transfers._approveFunds(routingToken, exchange, _totalRouting);
      IExchange(exchange).joinPoolFromInput(reserveToken, routingToken, _totalRouting, 1);
  }
  uint256 _totalBalance = Transfers._getBalance(reserveToken);
  require(_totalBalance >= _minRewardAmount, "high slippage");
  _deposit(_totalBalance);
}
```

The exchange parameter can be changed by the owner to the malicious address that snatches tokens. The owner will then call gulp with the _minRewardAmount==0 and all rewards will be taken. This attack is also possible with fee collectors or the buyback contract.

Recommendation

To avoid any sudden changes to the parameters, use a timelock.

6.2 | New deposits are instantly getting a share of undistributed rewards Major

Description

The current pending rewards cannot be withdrawn or re-invested until a new deposit is made. They are not included in the calculation of the number shares the depositor will receive. The number of shares is calculated like there are no pending rewards. This is not the only problem. Withdrawals are occurring without taking into account the pending rewards. It is more sensible to withdraw the reward right after you have taken a sip. This issue is not only unfair in the distribution of rewards during deposit/withdrawal but also creates an attack vector.

The Attack

The rewards of the gulp function are distributed equally across all current deposits, even those that were made just recently, if the deposit is made before the gulp function is called. So if the deposit-gulp-withdraw sequence is executed, the caller receives guaranteed profit. The attacker can also execute these functions quickly (in one transaction or block) and borrow a large amount of tokens to deposit the tokens. This will take almost all the rewards out of the gulp. An easy flashloan attack that involves one transaction can be executed by the owner, miner or whitelisted contract. This is possible even if the onlyEOAorWhitelist modifiable stops working or becomes disabled (issue 6.9). Anyone can make an attack even if onlyEOAorWhitelist works properly. Because there is no price manipulation, the risk is minimal. The attack will not result in price manipulation, but the price of the attack will be the same (a few blocks maximum).

Recommendation

If issue number 6.3 can be fixed without allowing anyone to call the gulp contract then the best solution is to include the gulp call in the deposit. Then withdraw. If you withdraw, it should be possible to not call gulp in an emergency situation.

6.3 | Proactive sandwiching of the gulp calls Major

Description

Each strategy token contract offers a way to retrieve pending rewards, convert them into reserve tokens and split the remaining balances. One share is paid to the fee collector to collect a performance fee. The rest is deposited to MasterChef contracts to earn more rewards. Passing a minimum slippage value to the function call prevents suboptimal trades. If the trade(s) does not provide the required reserve token amount, revert is issued.

The slippage parameter, and trades in gulp, open the function to proactive sandwich attacks. An attacker can set the slippage parameter to make arbitrarily poor trades. This is based on the extent that he can manipulate the liquidity around the gulp call.

The following assumptions make this attack vector significant:

- Trades on the exchange where they are performed allow significant changes in liquidity pools to be made in one transaction.
- With reasonable slippage, the attacker can outrun legitimate `gulp` calls.
- Trades are made, i.e. When `rewardToken!= routingToken`, and/or `routingToken!= reserveToken` are true.

Examples

This has an impact on the `gulp` functions of all strategies.

- `PancakeSwapCompoundingStrategyToken`
- `AutoFarmCompoundingStrategyToken`
- `PantherSwapCompoundingStrategyToken`

Moreover, fees collectors as well as buyback adapters:

- `PantherSwapBuybackAdapter`
- `AutoFarmFeeCollectorAdapter`
- `PancakeSwapFeeCollector`
- `UniversalBuyback`

Recommendations

There are many possible solutions and each one has its own tradeoffs. The following was our initial suggestion:

To ensure that only authorized parties have reasonable slippages are able to execute trades for the strategy contracts, the `onlyOwner` modifier must be added to the function. Additional slippage checks may be added to prevent unintentional behavior by authorized addresses. For example, to stop a bot from setting unreasonable slippage values due a software bug.

We came up with an alternative solution to address issue 6.2.

You can use oracles to prevent users calling the `gulp` function with excessive slippage (more that 5% of the oracle's moving-average price). This solution has the side effect that it will sometimes be used. This means that nobody will be able call the `gulp` when the price crashes.

6.4 | Expected amounts of tokens in the withdraw function Medium

Description

Each withdraw function in the strategy contract calculates the expected amount for the returned tokens prior to withdrawing them:

wheat-v1-core-audit/contracts/PantherSwapCompoundingStrategyToken.sol:L200-L208

```
function withdraw(uint256 _shares, uint256 _minAmount) external onlyEOAorWhitelist nonReentrant
{
        address _from = msg.sender;
        (uint256 _amount, uint256 _withdrawalAmount, uint256 _netAmount) = _calcAmountFromShares(_shares);
        require(_netAmount >= _minAmount, "high slippage");
        _burn(_from, _shares);
        _withdraw(_amount);
        Transfers._pushFunds(reserveToken, _from, _withdrawalAmount);
}
```

The contract then attempts to transfer the pre-calculated amount the msg.sender. It is not possible to verify that the amount intended was transferred to the strategy contract. Lower amounts may cause the withdrawal function to be reverted and tokens locked up.

Although we didn't find any case in which tokens were returned at a different amount, it's still a smart idea to do so to reduce reliance on external contracts.

Recommendation

There is a variety of options to address the problem:

Double-check the balance differences before and after MasterChef's withdrawal function is called.

This situation can be handled in emergency mode (issue 6.5)

6.5 | Emergency mode of the MasterChef contracts is not supported Medium

Description

All MasterChef contracts have an emergency withdrawal mode that allows for easier withdrawals (withdrawal of rewards).

```
// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public nonReentrant {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    uint256 amount = user.amount;
    user.amount = 0;
    user.rewardDebt = 0;
    user.rewardLockedUp = 0;
    user.nextHarvestUntil = 0;
    pool.lpToken.safeTransfer(address(msg.sender), amount);
    emit EmergencyWithdraw(msg.sender, _pid, amount);
}
```

```
// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    pool.lpToken.safeTransfer(address(msg.sender), user.amount);
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
    user.amount = 0;
    user.rewardDebt = 0;
}
```

```
// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public nonReentrant {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    uint256 wantLockedTotal =
        IStrategy(poolInfo[_pid].strat).wantLockedTotal();
    uint256 sharesTotal = IStrategy(poolInfo[_pid].strat).sharesTotal();
    uint256 amount = user.shares.mul(wantLockedTotal).div(sharesTotal);

    IStrategy(poolInfo[_pid].strat).withdraw(msg.sender, amount);

    pool.want.safeTransfer(address(msg.sender), amount);
    emit EmergencyWithdraw(msg.sender, _pid, amount);
    user.shares = 0;
    user.rewardDebt = 0;
}
```

Although it is difficult to predict when and how the emergency mode will be activated in MasterChef contracts, it is safer to have them available. The funds in the strategy contract will remain locked forever if there is an emergency.

Recommendation

Add the emergency mode implementation.

6.6 | The capping mechanism for Panther token leads = to increased fees Medium

Description

The transfer limit for Panther token is set at a maximum of 500 ml:

wheat-v1-core-audit/contracts/PantherSwapCompoundingStrategyToken.sol:L218-L245

```
function gulp(uint256 _minRewardAmount) external onlyEOAorWhitelist nonReentrant
{
	uint256 _pendingReward = _getPendingReward();
	if (_pendingReward > 0) {
		_withdraw(0);
	}
	uint256 __totalReward = Transfers._getBalance(rewardToken);
	(uint256 _feeReward, uint256 _retainedReward) = _capFeeAmount(__totalReward.mul(performanceFee) / 1e18);
	Transfers._pushFunds(rewardToken, buyback, _feeReward);
	if (rewardToken != routingToken) {
		require(exchange != address(0), "exchange not set");
		uint256 _totalReward = Transfers._getBalance(rewardToken);
		_totalReward = _capTransferAmount(rewardToken, _totalReward, _retainedReward);
		Transfers._approveFunds(rewardToken, exchange, _totalReward);
		IExchange(exchange).convertFundsFromInput(rewardToken, routingToken, _totalReward, 1);
	}
	if (routingToken != reserveToken) {
		require(exchange != address(0), "exchange not set");
		uint256 _totalRouting = Transfers._getBalance(routingToken);
		_totalRouting = _capTransferAmount(routingToken, _totalRouting, _retainedReward);
		Transfers._approveFunds(routingToken, exchange, _totalRouting);
		IExchange(exchange).joinPoolFromInput(reserveToken, routingToken, _totalRouting, 1);
	}
	uint256 _totalBalance = Transfers._getBalance(reserveToken);
	_totalBalance = _capTransferAmount(reserveToken, _totalBalance, _retainedReward);
	require(_totalBalance >= _minRewardAmount, "high slippage");
	_deposit(_totalBalance);
}
```

Recommendations

It is best to first cap __totalReward and then calculate fees based on that value.

6.7 | The _capFeeAmount function is not working as intended Medium

Description

Transfer size for Panther token is limited. Because of that, all the Panther transfer values in the PantherSwapCompoundingStrategyToken are also capped beforehand. To limit the fees, use the following function:

wheat-v1-core-audit/contracts/PantherSwapCompoundingStrategyToken.sol:L357-L366

```
function _capFeeAmount(uint256 _amount) internal view returns (uint256 _capped, uint256 _retained)
{
	_retained = 0;
	uint256 _limit = _calcMaxRewardTransferAmount();
	if (_amount > _limit) {
		_amount = _limit;
		_retained = _amount.sub(_limit);
	}
	return (_amount, _retained);
}
```

This function should return both the capped amount as well as the number of retained tokens. The _retained amount will always be 0.

Recommendations

Before changing the amount, calculate the retained value.

6.8 | Stale split ratios in UniversalBuyback Medium

Description

The gulp and pendingBurning functions of the UniversalBuyback contract use the hardcoded, constant values of DEFAULT_REWARD_BUYBACK1_SHARE and DEFAULT_REWARD_BUYBACK2_SHARE to determine the ratio the trade value is split with.

Any call to setRewardSplit for a new reward ratio will not work, but it will still emit a ChangeRewardSplit Event. This event may deceive system users and operators as it doesn't reflect the correct contract values.

Example

wheat-v1-core-audit/contracts/PantherSwapCompoundingStrategyToken.sol:L357-L366

```
uint256 _amount1 = _balance.mul(DEFAULT_REWARD_BUYBACK1_SHARE) / 1e18;
uint256 _amount2 = _balance.mul(DEFAULT_REWARD_BUYBACK2_SHARE) / 1e18;
```

wheat-v1-core-audit/contracts/PantherSwapCompoundingStrategyToken.sol:L357-L366

```
uint256 _amount1 = _balance.mul(DEFAULT_REWARD_BUYBACK1_SHARE) / 1e18;
uint256 _amount2 = _balance.mul(DEFAULT_REWARD_BUYBACK2_SHARE) / 1e18;
```

Recommendation

You should use `rewardBuyback1Share` or `rewardBuyback2Share` instead of the default values.

6.9 | Future-proofness of the `onlyEOAorWhitelist` modifier   Medium

Description

You will find the onlyEOAorWhitelist modifier in many places throughout the code. It checks that the transaction origin and message sender are equal in order to determine if the calling party is not a smart contractor.

If EIP-3074 is deployed with its AUTH, AUTHCALL and AUTH opcodes, this approach could not work.

The EOA check is dependent on tx.origin but the OpenZeppelin Reentrancy Guard does not. This can lead to additional attack vectors like flash loans. It is important to note that smart contracts that are not compatible with the protocol can limit their opportunities. Smart contracts cannot be integrated with GrowthDeFi in the same manner that GrowthDeFi does with third-party service providers.

Because it doesn't allow flash loan attacks by most users, the onlyEOAorWhitelist modifier can give you a false sense security. However, the same attack could still be carried out by certain people or at greater risk.

Modifiers do not affect whitelisted contracts or the owner.
You can disable the modifier:

wheat-v1-core-audit/contracts/WhitelistGuard.sol:L21-L28

```
modifier onlyEOAorWhitelist()
{
  if (enabled) {
      address _from = _msgSender();
      require(tx.origin == _from || whitelist.contains(_from), "access denied");
  }
  _;
}
```

This modifier is disallowed in the deployment script for testing purposes. It's important to remember to submit it on the production:

wheat-v1-core-audit/migrations/02_deploy_contracts.js:L50

```
await pancakeSwapFeeCollector.setWhitelistEnabled(false); // allows testing
```

Splitting the attack into multiple transactions is possible. Miners have the option to combine these transactions and take no additional risk. Regular users have the option to take a chance, borrow the money, and then execute the attack in multiple transactions, or even blocks.

Recommendation

It is highly recommended that you monitor the progress of the EIP and the potential implementation on Binance Smart Chain. The development team should make sure that the contract system is updated to reflect this new functionality. We recommend that you less rely on the fact only EOA can call the functions. It is much better to write code that can be called externally by smart contracts without compromising its security.

6.10 | Exchange owner might steal users' funds using reentrancy

Medium

Description

SafeTransferFrom is a method of removing funds from users and later pushing some of those funds back to them. This can be done in several places within the Exchange contract. In case one of the used token contracts (or one of its dependent calls) externally calls the Exchange owner, the owner may utilize that to call back `Exchange.recoverLostFunds` and drain (some) user funds.

Examples

wheat-v1-core-audit/contracts/Exchange.sol:L80-L89

```
function convertFundsFromInput(address _from, address _to, uint256 _inputAmount, uint256 _minOutputAmount) external override returns (uint256 _outputAmount)
{
	address _sender = msg.sender;
	Transfers._pullFunds(_from, _sender, _inputAmount);
	_inputAmount = Math._min(_inputAmount, Transfers._getBalance(_from)); // deals with potential transfer tax
	_outputAmount = UniswapV2ExchangeAbstraction._convertFundsFromInput(router, _from, _to, _inputAmount, _minOutputAmount);
	_outputAmount = Math._min(_outputAmount, Transfers._getBalance(_to)); // deals with potential transfer tax
	Transfers._pushFunds(_to, _sender, _outputAmount);
	return _outputAmount;
}
```

wheat-v1-core-audit/contracts/Exchange.sol:L121-L130

```
function joinPoolFromInput(address _pool, address _token, uint256 _inputAmount, uint256 _minOutputShares) external override returns (uint256 _outputShares)
{
	address _sender = msg.sender;
	Transfers._pullFunds(_token, _sender, _inputAmount);
	_inputAmount = Math._min(_inputAmount, Transfers._getBalance(_token)); // deals with potential transfer tax
	_outputShares = UniswapV2LiquidityPoolAbstraction._joinPoolFromInput(router, _pool, _token, _inputAmount, _minOutputShares);
	_outputShares = Math._min(_outputShares, Transfers._getBalance(_pool)); // deals with potential transfer tax
	Transfers._pushFunds(_pool, _sender, _outputShares);
	return _outputShares;
}
```

wheat-v1-core-audit/contracts/Exchange.sol:L99-L111

```
function convertFundsFromOutput(address _from, address _to, uint256 _outputAmount, uint256 _maxInputAmount) external override returns (uint256 _inputAmount)
{
	address _sender = msg.sender;
	Transfers._pullFunds(_from, _sender, _maxInputAmount);
	_maxInputAmount = Math._min(_maxInputAmount, Transfers._getBalance(_from)); // deals with potential transfer tax
	_inputAmount = UniswapV2ExchangeAbstraction._convertFundsFromOutput(router, _from, _to, _outputAmount, _maxInputAmount);
	uint256 _refundAmount = _maxInputAmount - _inputAmount;
	_refundAmount = Math._min(_refundAmount, Transfers._getBalance(_from)); // deals with potential transfer tax
	Transfers._pushFunds(_from, _sender, _refundAmount);
	_outputAmount = Math._min(_outputAmount, Transfers._getBalance(_to)); // deals with potential transfer tax
	Transfers._pushFunds(_to, _sender, _outputAmount);
	return _inputAmount;
}
```

wheat-v1-core-audit/contracts/Exchange.sol:L139-L143

```
function recoverLostFunds(address _token) external onlyOwner
{
	uint256 _balance = Transfers._getBalance(_token);
	Transfers._pushFunds(_token, treasury, _balance);
}
```

Recommendation

Reentrancy guard protection should be added to `Exchange.convertFundsFromInput`, `Exchange.convertFundsFromOutput`, `Exchange.joinPoolFromInput`, `Exchange.recoverLostFunds` at least, and in general to all public/external functions since gas price considerations are less relevant for contracts deployed on BSC.

APPENDIX 1 - FILES IN SCOPE

The following files were included in the audit:

File	SHA-1 hash
./AutoFarmCompoundingStrategyToken.sol	c682e1f7d6d0acfd26933ad3169dbbd2fdd4562c
./AutoFarmFeeCollectorAdapter.sol	41961d71d902acc31dcfd1274afa5a7f060ae671
./Exchange.sol	5d83f2881d5e3e2053fa96b97fcd53595e1192dc
./IExchange.sol	35e7cff12a28758d502ff452b512ef99959150e2
./interop/AutoFarmV2.sol	f2fd89fc8cfac68225af9cd47c36e5080255238a
./interop/Belt.sol	3ff6a4bcbe7eb59de56f144f476aa66337d69715
./interop/MasterChef.sol	7547c901e7068cf19dff4e8265dfb2669f06143d
./interop/PantherSwap.sol	83401998c19ae3c47fa01d61a6ea1a22c394a655
./interop/UniswapV2.sol	929d36dd4ec3b53364423b54098a076b5fefd85b
./interop/WrappedToken.sol	f735d7d325ac4b20e5447a532aced5d4f7c31b8a
./Migrations.sol	55bfb09493c7ecea45ed2ab9366db665af70aee2
./modules/Math.sol	2fcff034aba0c7dec9b7f5caae6295b21372871c
./modules/Transfers.sol	a7439175b42844b3b8ff7b593987a33fd6ceb3ee
./modules/UniswapV2ExchangeAbstraction.sol	cf58e61b9a4583cbb57fe34ef7a54b2f5237033f
./modules/UniswapV2LiquidityPool	f8e0e3dd5de61da29871b249f701279cbb23304c
Abstraction.sol	
./modules/Wrapping.sol	e02e0c9380dc3a281a5d6f43c0a9e5e39d854764
./network/$.sol	1d15880a1c99ba39e26020d40144bf7325f0b642
./PancakeSwapCompoundingStrategyToken.sol	b8841d52f589292bd5b5759977917a8846bcbad8
./PancakeSwapFeeCollector.sol	c303eaf8ff61b5c4dd8f45b2f7fa417422c790a9
./PantherSwapBuybackAdapter.sol	13449e0644b560640089c560aff4954ffd5177bf
./PantherSwapCompoundingStrategyToken.sol	e687c9f356c91a3933bbc4a3c74a065c57ed156e
./UniversalBuyback.sol	209a21322c45c9e1da92503d69fb794530a2dcd3
./WhitelistGuard.sol	d6fb7ddca4a85222e58196693306c75fbd777f42

APPENDIX 2 CLOSURE

ConsenSys Dialigence ("CD") receives compensation from clients (the Clients) for the analysis performed in these reports (the Reports). Reports can be distributed via ConsenSys publications or other distributions.

Reports are not intended to endorse or indict any project or team. They also do not guarantee security for any project. This Report doesn't consider or have any bearing on the economics of token sales, tokens, or any other product, services, or assets. Cryptographic tokens, which are emerging technologies, carry high technical risks and uncertainties. Any Report does not provide any representation or warranty to Third-Parties in any way. This includes regarding the bug-free nature of code, any business model or proprietors, or the legal compliance of such businesses. The Reports should not be relied upon by any third party, even if it is used to make decisions about buying or selling tokens, products, services, or assets. This Report is not intended as investment advice and should not be relied on as such. It is also not an endorsement of the project or its team. Furthermore, it does not guarantee absolute security. CD is not obligated to any Third-Party for publishing these Reports.

PURPOSE OF REPORTS Reports and analysis are only for Clients. They can be published with their permission. Our review is restricted to code reviews. We only review the code that we consider relevant for this report. Each Solidity code presents unique and unquantifiable risk because the Solidity language is still under development and can be subject to unknown risks. The review does NOT cover the compiler layer or any other code areas that may pose security risks. Cryptographic tokens, which are emerging technologies, carry high technical risk and uncertainty. Depending on the nature of the engagement, we might perform penetration testing and infrastructure assessments.

CD makes the Reports accessible to clients and other parties (i.e. "third parties") via its website. CD hopes that the public availability of these analyses will help the blockchain ecosystem to develop best practices in this rapidly changing area of innovation.

LINKS TO OTHER WEBSITES FROM THIS WEB site You can, via hypertext or other computer hyperlinks, gain access web sites owned by people other than ConsenSys. These hyperlinks are provided only for your convenience and are not intended to replace the owners of these web sites. ConsenSysys or CD are not responsible or liable for any content or operation of these Web sites. You also agree that ConsenSysys or CD will not be liable for any third-party Web site. Except as stated below, linking from this Web Site to another site does not mean or imply that ConsenSysys or CD endorses that site's content or its operator. It is up to you to decide whether or not you can use content from any other websites to which the Reports link. ConsenSys or CD will not be responsible for third-party software used on the Web Site. They also assume no liability for any errors or inaccuracies of any output generated by such software.

TIMELINESS CONTENT. The Reports are current as of the Report's date. However, they can be modified at any time. ConsenSys or CD are the only sources of information, unless otherwise indicated.