PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Preliminary Report

For BooXmas

11 December 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.

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 BooXmas on the Fantom 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	BooXmas
URL	https://xmas-past.com
Platform	Fantom
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`Token`	`0xD3111Fb8BDf936B11fFC9eba3b597BeA21e72724`	MATCH
`MasterChef`	`0x138c4dB5D4Ab76556769e4ea09Bce1D452c2996F`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	2	1	1	-
Low	1	1	-	-
Informational	10	9	-	1
Total	**16**	**14**	**1**	**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 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 Token

ID	Severity	Summary	Status
01	HIGH	`updateSwapRouter` could be used to siphon all liquidity fees or turn the token into a honeypot	RESOLVED
02	HIGH	LP tokens from addLiquidity are sent to lpEarner	RESOLVED
03	HIGH	`payoutToken` allow operator to drawing Xpast tokens used for liquidity from the token contract	RESOLVED
04	MEDIUM	Sensitive state variable modifying functions can be changed instantaneously	PARTIAL
05	MEDIUM	Masterchef needs to be whitelisted in `_excludedFromAntiWhale` or large harvest amounts will revert	RESOLVED
06	LOW	Inefficient usage of single address variables for whitelisting	RESOLVED
07	INFO	Lack of events for functions that change sensitive variables	RESOLVED
08	INFO	Typographical error for `burnAddr`	RESOLVED
09	INFO	`liqFee` and `liqFeeUpdated` are unused	RESOLVED
10	INFO	Unnecessary setting of amount to `sendAmount` in internal transfer function	RESOLVED
11	INFO	`feeAddr` can be constant	RESOLVED
12	INFO	Comments contradict code regarding fees	RESOLVED
13	INFO	Comments mention BNB when the project will be deployed on the Fantom Opera network and use FTM	RESOLVED

1.3.2 MasterChef

ID	Severity	Summary	Status
14	INFO	`pendingXpast` will show inaccurate pending harvests on the dapp frontend if the pending rewards causes `totalSupply` to exceed the token max supply	RESOLVED
15	INFO	Gas optimization by caching `xpast.totalSupply()` and `xpast.maxSupply()` as local variables	ACKNOWLEDGED
16	INFO	`MAX_EMISSION_RATE` has too many digits	RESOLVED

2 Findings

2.1 Token

The Xpast token is an ERC-20 token which will be used as the main reward token for the Masterchef. It allows for Xpast tokens to be minted when the `mint` function is called by the owner of the contract, which at the time of deployment would be the BooXMas team. Ownership of the token is generally transferred to the Masterchef at the time of deployment. There is a maximum supply of 288,000 Xpast tokens.

There is a maximum amount of tokens per transaction. If a transfer greater than this amount is made, and the sender or recipient is not in `_excludedFromAntiWhale`, the transaction will revert and not successfully complete.

For each non-whitelisted transfer, a maximum fee of 20% is enforced, and this includes the burn fee and the liquidity fee. The fee percentages can be changed by the operator.

Whitelisted addresses are determined as the following:

- If recipient is the burn address
- If sender is operator
- If sender is one of the `noTaxSenderAddr`
- If recipient is one of the `noTaxRecipientAddr`

For the liquidity fee, it is used in swapAndLiquify when the contract has at least more Xpast tokens than `minAmountToLiquify`. Half of the `minAmountToLiquify` amount of Xpast tokens is swapped to FTM, and 20% of the received FTM is sent to a hardcoded EOA fee address. The remaining FTM is paired with `minAmountToLiquify` amount of Xpast tokens and added as liquidity. As 20% of

the FTM received is sent out to the fee address, all of the FTM received from the swap will be used, and there should not be any FTM dust in the token contract. For the burn fee, it is transferred to the burn address.

2.1.1 Token Overview

Address	TBC
Token Supply	288,000
Decimal Places	18
Transfer Max Size	Lower cap of 1% of supply
Transfer Min Size	No minimum
Transfer Fees	Up to 20%
Pre-mints	TBC

2.1.2 Privileges

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

- `transferOwnership`
- `renounceOwnership`
- `mint`

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

- `updateTransferTaxRate`
- `updateBurnFee`
- `updateMaxTransferAmountRate`
- `updateMinAmountToLiquify`
- `setExcludedFromAntiWhale`
- `updateSwapAndLiquifyEnabled`

- updateSwapRouter
- setNoTaxSenderAddr
- setNoTaxRecipientAddr
- setNoTaxSenderAddr1
- setNoTaxRecipientAddr1
- setNoTaxSenderAddr2
- setNoTaxRecipientAddr2
- setNoTaxSenderAddr3
- setNoTaxRecipientAddr3
- setNoTaxSenderAddr4
- setNoTaxRecipientAddr4
- setNoTaxSenderAddr5
- setNoTaxRecipientAddr5
- setNoTaxSenderAddr6
- setNoTaxRecipientAddr6
- setNoTaxSenderAddr7
- setNoTaxRecipientAddr7
- setNoTaxSenderAddr8
- setNoTaxRecipientAddr8
- setLpEarnerAddr
- payoutToken
- transferOperator

2.1.3 Issues & Recommendations

Issue #01	**`updateSwapRouter` could be used to siphon all liquidity fees or turn the token into a honeypot**
Severity	HIGH SEVERITY
Description	The operator can update the router that generates liquidity to an address or contract of choice. This contract could be a malicious contract that simply keeps the tokens sent to it or even worse a contract that presents selling transactions by always reverting.
Recommendation	Consider removing this function. If this is not possible, consider a significantly longer timelock as the operator so that users can have sufficient time to react to future router changes.
Resolution	RESOLVED Added `require (address(swapRouter) == address(0), "Router already set");` to allow the setting of `swapRouter` only once.

Issue #02	LP tokens from `addLiquidity` are sent to `lpEarner`
Severity	HIGH SEVERITY

Location	Line 1108
	`function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {`
	`    // approve token transfer to cover all possible scenarios`
	`    _approve(address(this), address(swapRouter), tokenAmount);`
	
	`    // add the liquidity`
	`    swapRouter.addLiquidityETH{value: ethAmount}(`
	`        address(this),`
	`        tokenAmount,`
	`        0, // slippage is unavoidable`
	`        0, // slippage is unavoidable`
	`        viewLpEarner(),`
	`        block.timestamp`
	`    );`
	`}`

Description	The LP tokens provided from the `addLiquidity` are sent to the `lpEarner`, who will be able to decompose the LPs and obtain the underlying tokens.
Recommendation	Consider setting the `to` address for adding of liquidity to a burn address, ensuring that the received liquidity pool tokens are locked forever.
Resolution	RESOLVED `lpEarner` is now set to the burn address. `address constant public lpEarner = 0x000000000000000000000000000000000000dEaD;`

Issue #03	**payoutToken allow operator to drawing Xpast tokens used for liquidity from the token contract**
Severity	HIGH SEVERITY
Description	`payoutToken` allows the transfer of any ERC20 token in the token contract to the operator. As there will be Xpast tokens from the liquidity tax in the contract, there will be a significant amount of tokens which can be withdrawn.
Recommendation	Consider removing this function. If this is not possible, consider adding a check to only allow all tokens to be transferred except the Xpast token. `require(address(this) != _token, "No Xpast token");`
Resolution	RESOLVED `payoutToken` has now been removed.

Issue #04	Sensitive state variable modifying functions can be changed instantaneously
Severity	MEDIUM SEVERITY
Description	The following functions can be called to change state variables without any timelock: - `updateTransferTaxRate` - `updateBurnFee` - `updateMaxTransferAmountRate` - `updateMinAmountToLiquify` - `setExcludedFromAntiWhale` - `updateSwapAndLiquifyEnabled` - `updateSwapRouter` - `setNoTaxSenderAddr` - `setNoTaxRecipientAddr` - `setNoTaxSenderAddr1` - `setNoTaxRecipientAddr1` - `setNoTaxSenderAddr2` - `setNoTaxRecipientAddr2` - `setNoTaxSenderAddr3` - `setNoTaxRecipientAddr3` - `setNoTaxSenderAddr4` - `setNoTaxRecipientAddr4` - `setNoTaxSenderAddr5` - `setNoTaxRecipientAddr5` - `setNoTaxSenderAddr6` - `setNoTaxRecipientAddr6` - `setNoTaxSenderAddr7` - `setNoTaxRecipientAddr7` - `setNoTaxSenderAddr8` - `setNoTaxRecipientAddr8` - `setLpEarnerAddr` - `payoutToken` Each of these functions can result in behavior that differs from expectations on how the token contract should behave.
Recommendation	The operator of the contract should be set to a reasonably delayed timelock to allow users to react to any upcoming changes.
Resolution	PARTIALLY RESOLVED The time has placed the operator behind a 2-hour timelock. This issue remains partially resolved as we do not feel it is sufficient.

Issue #05	**Masterchef needs to be whitelisted in `_excludedFromAntiWhale` or large harvest amounts will revert**
Severity	MEDIUM SEVERITY
Description	If Masterchef is not whitelisted in `_excludedFromAntiWhale` and a harvest amount which exceeds `maxTransferAmount` is done, the transaction will revert. This would result in the user being unable to make normal deposit or withdrawals from the Masterchef.
Recommendation	Ensure that the Masterchef contract address is whitelisted in `_excludedFromAntiWhale`.
Resolution	RESOLVED The Masterchef has been whitelisted in `_excludedFromAntiWhale`.

Issue #06	Inefficient usage of single address variables for whitelisting
Severity	LOW SEVERITY
Location	Line 1434 (example) `if (recipient == brunAddr || sender == _operator || sender == noTaxSenderAddr || recipient == noTaxRecipientAddr || sender == noTaxSenderAddr1 || recipient == noTaxRecipientAddr1 || sender == noTaxSenderAddr2 || recipient == noTaxRecipientAddr2 || sender == noTaxSenderAddr3 || recipient == noTaxRecipientAddr3 || sender == noTaxSenderAddr4 || recipient == noTaxRecipientAddr4 || sender == noTaxSenderAddr5 || recipient == noTaxRecipientAddr5 || sender == noTaxSenderAddr6 || recipient == noTaxRecipientAddr6 || sender == noTaxSenderAddr7 || recipient == noTaxRecipientAddr7 || sender == noTaxSenderAddr8 || recipient == noTaxRecipientAddr8 || transferTaxRate == 0) {`
Description	Instead of mapping to whitelist `noTaxRecipientAddr` and `noTaxSenderAddr`, 9 single address state variables are used each. This results in a massive number of setter functions, each for a single state variable, and conditionals in the transfer statement.
Recommendation	Consider switching to 2 mappings: one for `noTaxRecipient` and another for `noTaxSender`. This can reduce the setters to one for each mapping, and the if statement to the following: `if (recipient == burnAddr || sender == _operator || noTaxRecipient[recipient] || noTaxSender[sender] || transferTaxRate == 0 ) {`
Resolution	RESOLVED Mappings are now used.

Issue #07	Lack of events for functions that change sensitive variables
Severity	LOW SEVERITY
Description	Functions that affect the status of sensitive variables should emit events. - `setExcludedFromAntiWhale` - `payoutToken`
Recommendation	Add events for the setter functions which modify sensitive state variables.
Resolution	RESOLVED Event is emitted for `setExcludedFromAntiWhale`. `payoutToken` has been removed.

Issue #08	Typographical error for burnAddr
Severity	INFORMATIONAL
Description	The `burnAddr` variable is mis-spelled as `brunAddr`.
Recommendation	Consider correcting the typographical error.
Resolution	RESOLVED The error has been corrected.

Issue #09	`liqFee` and `LiqFeeUpdated` are unused
Severity	INFORMATIONAL
Description	As the actual liquidity tax amount is calculated based on the burn fee percentage, `liqFee` and `LiqFeeUpdated` are unused and unnecessary.
Recommendation	Remove the unnecessary state variable and event.
Resolution	RESOLVED `LiqFeeUpdated` has been removed.

Issue #10	Unnecessary setting of amount to sendAmount in internal transfer function
Severity	INFORMATIONAL
Location	Line 1451 `amount = sendAmount;`
Description	In the last line of `_transfer`, `amount` is updated to `sendAmount`, but never used following that.
Recommendation	Remove the unnecessary statement.
Resolution	RESOLVED The unnecessary statement has been removed.

Issue #11	`feeAddr` can be constant
Severity	INFORMATIONAL
Description	`feeAddr` is defined once and never changed, so it can be set as constant to save gas.
Recommendation	Declare `feeAddr` as constant.
Resolution	RESOLVED `feeAddr` is now constant.

Issue #12	Comments contradict code regarding fees
Severity	INFORMATIONAL

Location	Line 1242
	`// Transfer tax rate in basis points. (default 4.5%)`
	`uint16 public transferTaxRate = 500;`
	
	Line 1251
	`// Max transfer tax rate: 10%.`
	`uint16 public constant maxTaxRate = 2000;`

Description	The following portions of the contract show that the comments and actual code contradict each other regarding the fees.
Recommendation	Either update the comments or code to reflect the actual default for `transferTaxRate`, and the amount of `maxTaxRate`.
Resolution	RESOLVED Comments have been updated to reflect the code.

Issue #13	Comments mention BNB when the project will be deployed on the Fantom Opera network and use FTM
Severity	INFORMATIONAL

Location	Line 1543
	`// To receive BNB from swapRouter when swapping`
	`receive() external payable {}`
	
	Line 1430
	`&& sender != swapPair  // excludes Buying in BNB Liquidity`

Description	The comments mention BNB, when the actual deployment will be on Phantom using FTM.
Recommendation	Update the comments to reflect the correct currency.
Resolution	RESOLVED Comments have been updated to reflect the correct currency.

2.2 MasterChef

The BooXMas Masterchef is a fork of Goose Finance's Masterchef. A notable feature of forking the latter is the removal of the `migrator` function from Sushiswap, which can possibly be used to steal user's tokens. BooXMas has limited the deposit fee to at most 4%. The emissions have been modified from per block to per second.

The Masterchef has a check that ensures that only a maximum of 288,000 Xpast tokens are minted. The maximum emission that can be set is 10 tokens per second.

2.2.1 Privileges

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

- `add`
- `set`
- `setFeeAddress`
- `setEmissionRate`
- `setStartBlock`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #14	**`pendingXpast` will show inaccurate pending harvests on the dapp frontend if the pending rewards causes `totalSupply` to exceed the token max supply**
Severity	INFORMATIONAL
Description	`pendingXpast` does not check if the pending rewards will cause the `totalSupply` to exceed the token `maxSupply`. This can cause inaccurate pending harvests to be shown towards the end of token emissions.
Recommendation	Consider factoring in the token `maxSupply`, and set the pending reward to be the difference between token `maxSupply` and `totalSupply` if the pending reward causes `totalSupply` to exceed token `maxSupply`.

```
if ( block.timestamp > pool.lastRewardSecond &&
    pool.lpSupply != 0 &&
    totalAllocPoint > 0 ) {
    uint256 multiplier = getMultiplier(
        pool.lastRewardSecond,
        block.timestamp
    );
    uint256 xpastReward = multiplier
        .mul(XpastPerSecond)
        .mul(pool.allocPoint)
        .div(totalAllocPoint);

    uint256 devReward = xpastReward.div(10);
    uint256 totalRewards =
xpast.totalSupply().add(devReward).add(
        xpastReward
    );

    if (totalRewards > xpast.maxSupply()) {
        xpastReward =
xpast.maxSupply().sub(xpast.totalSupply());
    }

    accXpastPerShare = accXpastPerShare.add(
        xpastReward.mul(1e18).div(pool.lpSupply)
    );
}
```

Resolution

RESOLVED

The recommendation has been implemented.

Issue #15	**Gas optimization by caching `xpast.totalSupply()` and `xpast.maxSupply()` as local variables**
Severity	INFORMATIONAL
Description	As `xpast.totalSupply()` and `xpast.maxSupply()` is called multiple times in the `updatePool` function, gas usage can be optimized by setting the value returned by those function calls as local variables, and reusing that local variable within the function.

Recommendation	Set and use the following local variable in `updatePool`:
	`uint256 totalSupply = xpast.totalSupply();`
	`uint256 maxSupply = xpast.maxlSupply();`

Resolution	ACKNOWLEDGED

Issue #16	**`MAX_EMISSION_RATE` has too many digits**
Severity	INFORMATIONAL
Location	Line 953 `uint256 public constant MAX_EMISSION_RATE = 500000000000000000;`
Description	Literals with many digits are difficult to read and review.
Recommendation	Change the `MAX_EMISSION_RATE` to something more readable such as 5 ether.
Resolution	RESOLVED The `MAX_EMISSION_RATE` has been updated to 5 ether.


PALADIN
BLOCKCHAIN SECURITY