PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For MetaLove

18 March 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 MetaLove on the BNB Smart Chain (BSC). 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	MetaLove
URL	https://metalovebsc.com/
Platform	BNB Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
MasterChefMLC	0xEE740260e0709e47Fd9005Cd7518A4e530268AA8	MATCH
MasterChefMLG	0xCB0b3b803b86dE6a5A2f61268107E6163D9b4168	MATCH
MetaLoveCoinToken	0x63CB84FC6247d337c6b2654AC218C9c8cf1A7a60	MATCH
MetaLoveGoldToken	0x329840D252a324396C13B9c8E0f382155E4D7904	MATCH
Timelock	0x4F00bFbBe36A3173B4BB6CFC2577bb0679b19feF	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	2	-	-
Medium	0	-	-	-
Low	2	2	-	-
Informational	16	16	-	-
Total	**20**	**20**	**-**	**-**

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 MasterChefMLC and MasterChefMLG

ID	Severity	Summary	Status
01	HIGH	The contracts do not support fee-on-transfer tokens	RESOLVED
02	HIGH	emergencyWithdraw is vulnerable to reentrancy which could cause theft of any token which allows for reentrancy	RESOLVED
03	LOW	Emission rates are not adjustable by the owner	RESOLVED
04	INFO	Contract contains unused functionality	RESOLVED
05	INFO	Several variables can be made immutable	RESOLVED
06	INFO	Lack of validation	RESOLVED
07	INFO	Inconsistency: The initial native token pool is not added to poolExistence allowing governance to add the pool twice	RESOLVED
08	INFO	Pool uses the contract balance to figure out the total deposits	RESOLVED
09	INFO	The pendingReward and updatePool functions will revert if totalAllocPoint is zero	RESOLVED
10	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED
11	INFO	A user could deposit before startBlock to do an early harvest right after the project launches	RESOLVED
12	INFO	return variable is not checked on transfer	RESOLVED
13	INFO	Several variables can be made external	RESOLVED
14	INFO	Lack of events for add and set	RESOLVED
15	INFO	Typographical errors	RESOLVED

1.3.2 MetaLoveCoinToken and MetaLoveGoldToken

ID	Severity	Summary	Status
16	LOW	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	RESOLVED
17	INFO	Governance functionality is broken	RESOLVED
18	INFO	delegateBySig can be frontrun and cause denial of service	RESOLVED
19	INFO	mint can be made external	RESOLVED
20	INFO	The contract contains typographical errors	RESOLVED

1.3.3 Timelock

No issues found.

2 Findings

2.1 MasterChefMLC and MasterChefMLG

The MasterChefMLC and MasterChefMLG Masterchef contracts are modified forks of the Panther Masterchef. Just like Panther, rewards can only be harvested after a specified interval, which can be configured to a maximum of 4 hours, has passed. This Masterchef also improves upon traditional deposit fee-based Masterchefs by limiting the maximum deposit fee to 5%.

3% of all emissions are sent to the fee address as a reward for the team.

The contract has a referral mechanism which allows users to refer other users: 3% of each harvest is given as a bonus to the referrer in addition to the normal harvest. This bonus can be adjusted by the contract owner up to a maximum of 5% of each harvest.

During contract creation, the pool for the native token is immediately created with 1000 allocation points in weight. This pool is initially configured to not have a harvest interval so users will be able to harvest this pool at any time and as many times as they want. This pool also does not have a deposit fee.

This section of the report will exclusively refer to the MasterChefMLC contract. However, as this contract is identical to MasterChefMLG, all issues mentioned below equally apply to MasterChefMLG. The client should therefore fix them within both contracts for the issues to be marked as fully resolved.

The client has informed us that they have conducted manual testing on these contracts.

2.1.1 Privileged Roles

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

- `add`
- `set`
- `setFeeAddress`
- `updateReferralBonusBp`

2.1.2 Issues & Recommendations

Issue #01	The contracts do not support fee-on-transfer tokens
Severity	HIGH SEVERITY
Location	MasterChefMLC::263 `pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount)`
Description	If a fee-on-transfer token is added to any pool, it will result in the draining of the token balance as the Masterchef will credit more of the token since it uses the `amount` parameter supplied in the `deposit` function instead of actual received tokens.

Recommendation	Use the actual value received by the Masterchef instead of the amount specified by the user during the deposit.
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.safeTransferFrom(msg.sender, address(this), amount);`
	`amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`
	We would also like to remind the client that a before-after pattern like the one mentioned above always needs to be accompanied with a reentrancy guard since it is vulnerable to reentrancy attacks if the token transfer permits this.

Resolution	RESOLVED The before-after pattern has been introduced on both `deposit` functions.

Issue #02	**`emergencyWithdraw` is vulnerable to reentrancy which could cause theft of any token which allows for reentrancy**
Severity	HIGH SEVERITY
Location	Line 293 `function emergencyWithdraw(uint256 _pid) public {`
Description	The `emergencyWithdraw` function could allow reentrancy and steal tokens by calling the function multiple times until the contract is drained of that token. This is caused by the user information being updated after the tokens are sent to the users. If an exploiter were to call `emergencyWithdraw` again during the first token transfer, they would be able to withdraw the token twice.
Recommendation	Either adhere to checks-effects-interactions or add the `nonReentrant` modifier.
Resolution	RESOLVED The client has added a `nonReentrant` modifier.

Issue #03	Emission rates are not adjustable by the owner
Severity	LOW SEVERITY

Location	Line 126
	`function updateMultiplier(uint256 multiplierNumber) public onlyOwner {`
	`    BONUS_MULTIPLIER = multiplierNumber;`
	`}`

Description	The variable `BONUS_MULTIPLIER` is not used and there is no `setEmissionRate` function either. This prevents the owner from changing the actual emission rate.
Recommendation	Consider using the `BONUS_MULTIPLIER` or remove it entirely. Consider adding a `setEmissionRate` variable to change the actual emission rate. Either way, bounds should be added so that the emission rate cannot be set to a big number.
Resolution	RESOLVED The client removed `BONUS_MULTIPLIER` entirely. There is no way to change the actual emission rate once it has been set. An upper bound was added and set to 1 token per block, limiting the rate which can only be set during contract creation.

Issue #04	Contract contains unused functionality
Severity	INFORMATIONAL

Description	The contract contains sections of code which are not used. These can be confusing to third-party code reviewers and can make the code less accessible. The following sections of code can therefore be removed.
	
	Line 10
	`// import "@nomiclabs/buidler/console.sol"`
	
	Line 59
	`uint256 public BONUS_MULTIPLIER = 1`
	
	Line 87
	`mapping(IBEP20 => uint256) public poolIdForLpAddress;`
	
	Line 93
	`event EmissionRateUpdated(address indexed caller, uint256 previousAmount, uint256 newAmount);`
	
	Line 126
	`function updateMultiplier(uint256 multiplierNumber) public onlyOwner`

Recommendation	Consider removing the above lines of code in an effort to keep the contract as simple as possible.
Resolution	RESOLVED

Issue #05	Several variables 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: - `mlc` - `mlcPerBlock` - `startBlock` - `bonusEndBlock` This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly immutable.
Resolution	RESOLVED

Issue #06	Lack of validation
Severity	INFORMATIONAL

Description	The contract contains sections of code which lack proper validation. This could cause errors in case unexpected inputs are provided.
	
	Line 106
	`mlc = _mlc;`
	`require(address(_mlc) != address(0));`
	
	Line 107
	`feeAddress = _feeAddress;`
	`require(_feeAddress != address(0));`
	
	Line 108
	`mlcPerBlock = _mlcPerBlock;`
	`require(_mlcPerBlock <= MAX_EMISSION_RATE, "");`
	
	Line 109
	`startBlock = _startBlock;`
	`require(_startBlock > block.number, "");`
	
	Line 110
	`bonusEndBlock = _bonusEndBlock;`
	`require(_bonusEndBlock > _startBlock, "");`
	
	Line 168
	`poolInfo[_pid].allocPoint = _allocPoint;`
	`require(allocPoint <= MAX_ALLOC_POINT, "");`

Recommendation	Consider using the above requirements.
Resolution	RESOLVED

Issue #07	**Inconsistency: The initial native token pool is not added to `poolExistence` allowing governance to add the pool twice**
Severity	INFORMATIONAL

Location	Line 113
	`poolInfo.push(PoolInfo({`
	`lpToken: _mlc,`
	`allocPoint: 1000,`
	`lastRewardBlock: startBlock,`
	`accMLCPerShare: 0,`
	`depositFeeBP: 0,`
	`harvestInterval: 0`
	`}));`

Description	The initial native token pool is not added to `poolExistence` allowing governance to add the pool twice. This is inconsistent with the other pools within add.
Recommendation	Add the pool to `poolExistence`: `poolExistence[_mlc] = true;`
Resolution	RESOLVED

Issue #08	**Pool uses the contract balance to figure out the total deposits**
Severity	INFORMATIONAL
Description	As with pretty much all Masterchefs and staking contracts, the total number of tokens in the contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the Masterchef.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits.
Resolution	RESOLVED An `lpSupply` variable has been added to the pools.

Issue #09	**The `pendingReward` and `updatePool` functions will revert if `totalAllocPoint` is zero**
Severity	INFORMATIONAL
Location	Lines 195 & 228 `uint256 mlcReward = multiplier.mul(mlcPerBlock).mul(pool.allocPoint).div(totalAllocPoint);`
Description	In the `pendingReward` and `updatePool` functions, at some point a division is made by the `totalAllocPoint` variable. If all pools have their rewards set to zero, this variable will be zero as well. The requests will then revert with a division by zero error.

Recommendation	Consider only calculating the accumulated rewards since the `lastRewardTimestamp` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the block.timestamp and `lpSupply`, like so:
	
	Line 193
	`if (block.number > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint != 0) {`
	
	Line 223
	`if (lpSupply == 0 || totalAllocPoint == 0) {`

Resolution	RESOLVED

Issue #10	**Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `updatePool`, `deposit`, `withdraw` and the pending rewards function, `accMLCPerShare` is based upon the `lpSupply` variable. `accMLCPerShare = accMLCPerShare.add(mlcReward.mul(1e12).div(lpSupply));` However, if this `lpSupply` becomes a severely large value this will cause precision errors due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals.
Recommendation	Consider increasing precision to 1e24 across the entire contract. It should be noted that even a precision of 1e24 can be imprecise in some edge cases. In case the client thinks it is probable they will add tokens with a huge `totalSupply`, we recommend testing which precision variable is most appropriate to support them without potentially reverting due to overflows.
Resolution	RESOLVED The client has implemented our recommendation and has updated the precision multiplier to 1e24.

Issue #11	**A user could deposit before `startBlock` to do an early harvest right after the project launches**
Severity	INFORMATIONAL

Location	Line 307
	`if (user.nextHarvestUntil == 0) {`
	`user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval);`
	`}`

Description	A user that deposits before `startBlock` would be able to do an early harvest because `nextHarvestUntil` was set to a timestamp potentially close after or before the `startBlock`. This might surprise other users as some of the early stakers can do an early harvest.
Recommendation	Consider whether this poses a threat to the tokenomics of the project, and if so, consider accounting for it by not allowing harvests to occur up to a harvest interval after the farm has started.
Resolution	RESOLVED The client has added a minimum waiting delay of 4800 blocks after the project has launched.

Issue #12	`return` variable is not checked on `transfer`
Severity	INFORMATIONAL

Location	Line 333
	`mlc.transfer(_to, mlcBalance);`
	
	Line 335
	`mlc.transfer(_to, _amount);`

Description	The return variable of an ERC-20 tokens is not checked on `transfer`—this could be an issue if the contract is ever forked with a different token.
Recommendation	Consider using `safeTransfer` from OpenZeppelin.
Resolution	RESOLVED

Issue #13	Several variables 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: - `add` - `set` - `deposit` - `withdraw` - `emergencyWithdraw` - `setFeeAddress` - `updateReferralBonusBp` 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 above variables as external.
Resolution	RESOLVED

Issue #14	Lack of events for add and set
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. Inside constructor, the contract adds the native token as a pool, it should emit an event that the pool was created similarly to the one that should be emitted in `add`.
Recommendation	Consider adding events for the above functions.
Resolution	RESOLVED

Issue #15	Typographical errors
Severity	INFORMATIONAL
Description	The contract contains typographical errors on the following lines of code.

Line 9

```
import './MetaLoveCoinToken.sol';
```

The MetaLoveCoinToken does not need to be imported. Importing an interface with the appropriate mint function suffices.

Line 60

```
// Max harvest interval: 1 days.
```

The comment mentions that the max harvest interval is 1 day, while the max is set to 4 hours.

Line 72

```
uint256 public bonusEndBlock;
```

This is actually the real emissions end block as there are no bonus emissions.

Line 116

```
lastRewardBlock: startBlock,
```

This is inconsistent with lpToken where the local _mlc is used. Consider using _startBlock instead to promote consistency and furthermore save gas.

Line 142

```
require(_depositFeeBP <= MAXIMUM_DEPOSIT_FEE_BP, "set:
invalid deposit fee basis points");
```

This comment should mention add.

Line 163

```
require(_harvestInterval <= MAXIMUM_HARVEST_INTERVAL, "add:
invalid harvest interval");
```

This comment should mention `set`.

Lines 242, 263, 286, 296 and 344

```
pool.lpToken.safeTransferFrom(address(msg.sender),
address(this), _amount);
```

Casting `msg.sender` to address is unnecessary. This should be changed throughout the contract.

Line 352

```
emit Referral(_user, _referrer);
```

The parameters are emitted in the wrong order. The event can furthermore be called `ReferralSet`.

Line 357

```
function getReferral(address _user) public view returns
(address)
```

The address referring to someone is called the referrer, while the `_user` is the referral. Consider renaming this to `getReferrer`.

Lines 373 & 374

```
require(_newRefBonusBp <= MAXIMUM_REFERRAL_BP,
"updateRefBonusPercent: invalid referral bonus basis
points");
require(_newRefBonusBp != refBonusBP,
"updateRefBonusPercent: same bonus bp set");
```

The function is called `updateReferralBonusBp`.

Recommendation Consider fixing the above typographical errors.

Resolution RESOLVED

2.2 MetaLoveCoinToken and MetaLoveGoldToken

The MetaLoveGoldToken and MetaLoveCoinToken tokens are simple ERC-20 tokens which will be used as the main reward tokens for the two Masterchefs. They are nearly identical to each other and have therefore been aggregated into a single section. Each issue is present and should be resolved within both contracts.

These contracts allow for the tokens to be minted when the `mint` function is called by the owner of the token contract, which at the time of deployment would be the MetaLove team. Users should therefore carefully inspect that ownership has been transferred to their respective MasterChefs after deployment.

The client has informed us that they have conducted manual testing on these contracts.

2.2.1 Privileged Roles

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

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

2.2.2 Token Overview

Name	Meta Love Coin
Symbol	MLC
Address	0x63CB84FC6247d337c6b2654AC218C9c8cf1A7a60
Token Supply	7,560,000 (enforced in Masterchef)
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	500

Name	Meta Love Gold
Symbol	MLG
Address	0x329840D252a324396C13B9c8E0f382155E4D7904
Token Supply	10,800 (enforced in Masterchef)
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	500

2.2.3 Issues & Recommendations

Issue #16	**`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef**
Severity	LOW SEVERITY
Description	The `mint` function allows the owner (contract deployer) to mint tokens before ownership is transferred to the Masterchef. This could be used to mint a large amount of tokens and potentially dump them on user generated liquidity when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this `mint` function is to be used by letting your community know how much was minted, where the tokens are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	RESOLVED 500 tokens of each MLC and MLG contract have been pre-minted and ownership of the contracts has been transferred to their respective Masterchefs.

Issue #17	Governance functionality is broken
Severity	INFORMATIONAL
Description	Although there is YAM-related delegation code in the token contract which is usually used for governance and voting, the delegation code can be abused as the delegates are not moved during transfers and burns. This allows for double spending attacks on the voting mechanism. It should be noted that this issue is present in pretty much every single farm out there including PancakeSwap and even SushiSwap but it does render this whole mechanism useless. It is because of this reason that project like SushiSwap and PancakeSwap all use snapshot.org currently.
Recommendation	The broken delegation-related code can be removed to reduce the size of the contract. If voting is ever desired, it can still be done through snapshot.org, used by many of the larger projects.
Resolution	RESOLVED The client has deleted all delegation-related logic.

Issue #18	`delegateBySig` can be frontrun and cause denial of service
Severity	INFORMATIONAL
Description	Currently if `delegateBySig` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `delegateBySig` transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example.
Recommendation	Similar to the broken governance functionality issue, the delegate logic can just be removed.
Resolution	RESOLVED The client has deleted all delegation-related logic.

Issue #19	mint 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 function as external.
Resolution	RESOLVED

Issue #20	The contract contains typographical errors
Severity	INFORMATIONAL
Locations	Lines 115, 116, 117, 147, 184 and 220
Description	`CGIRL` / `EGIRL` references need to be adjusted to the actual name of the tokens.
Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED These sections have been removed from the report.

2.3 Timelock

The Timelock contract is a clean fork of Compound Finance's timelock. This is the most common contract used in DeFi to time lock governance access and is thus compatible with most third-party tools.

Parameter	Value	Description
Delay	6 hours	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	6 hours	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

2.3.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY