PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Space X

10 September 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 Space X on the Binance Smart Chain. 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	Space X
URL	https://farm.space/
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`SpaceXToken`	`SpaceXToken.sol`	PENDING
`SpaceXChef`	`SpaceXChef.sol`	PENDING

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	2	-	-
Medium	3	3	-	-
Low	3	2	-	1
Informational	6	5	-	1
Total	**14**	**12**	**0**	**2**

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 SpaceXToken

ID	Severity	Summary	Status
01	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	ACKNOWLEDGED

1.3.2 SpaceXChef

ID	Severity	Summary	Status
02	HIGH	`unimexFactory` retains infinite approval even after `unimexIntergration` is set to `False`	RESOLVED
03	HIGH	Funds can be stolen if `unimexFactory` is malicious	RESOLVED
04	MEDIUM	Contract makes external calls to an unknown contract	RESOLVED
05	MEDIUM	Transfers to the `unimexFactory` may not account for tokens with transfer taxes	RESOLVED
06	MEDIUM	Contract balance may be miscalculated if WETH is deposited to `unimexFactory`	RESOLVED
07	LOW	The `pendingSpaceX` function will revert if `totalAllocPoint` is zero	RESOLVED
08	LOW	Deposit fees may accrue as dust in the Masterchef	RESOLVED
09	INFO	`spaceXToken`, `unimexFactory`, `WETH` can be made immutable	RESOLVED
10	INFO	`unimexFactory` can be hard-coded rather than set in constructor	RESOLVED
11	INFO	Pools use the contract balance to figure out the total deposits	ACKNOWLEDGED
12	INFO	There are no sanity checks in the `setReferralAddress` function	RESOLVED
13	INFO	`safeWethTransfer` lacks non-zero safeguard	RESOLVED
14	INFO	Lack of events for `updateStartBlock`, `setReferralCommissionRate`, `set` and `add`	RESOLVED

2 Findings

2.1 SpaceXToken

The contract allows for SpaceX tokens to be minted when the `mint` function is called by the Owner, who at the time of deployment would be the deployer. However, ownership is generally transferred to the Masterchef via the `transferOwnership` function for emission rewards to be minted and distributed to users staking in the Masterchef.

The `mint` function can be used to pre-mint tokens for various uses including injection of initial liquidity, token presale, airdrops, and others.

2.1.1 Token Overview

Address	TBC
Token Supply	TBC
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	TBD

2.1.2 Privileged Roles

The owner of the SpaceXToken contract should be transferred to the Masterchef.

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

- `mint`

2.1.3 Issues & Recommendations

Issue #01	**`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 could be used to pre-mint tokens for legitimate uses including, but not limited to, the injection of initial liquidity, token presale, or airdrops; however, this function may also be used to pre-mint and dump tokens 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 they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	ACKNOWLEDGED Once the contract has been deployed and ownership transferred to the Masterchef, we will mark this issue as Resolved.

2.2 SpaceXChef

The Masterchef is a fork of PantherSwap's Masterchef. A notable feature of forking this Masterchef is the removal of the migrator function from Pancakeswap, which of late has been used maliciously to steal users' tokens. We commend Space X on their decision to fork a relatively safer version of the Masterchef. Finally, the deposit fees have an upper limit of 10%, which is a reasonable cap.

The referral mechanism gives the referral a 2% (can be set to a maximum of 5%) referral commission of earned rewards. It should be noted that the rewards are not take out of the harvests, but rather is minted directly to the referral.

2.2.1 Privileged Roles

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

- `add`
- `set`
- `setDevAddress`
- `setFeeAddress`
- `setVaultAddress`
- `updateEmissionRate`
- `setReferralAddress`
- `setReferralCommissionRate`
- `updateStartBlock`

2.2.2 Issues & Recommendations

Issue #02	**unimexFactory retains infinite approval even after unimexIntergration is set to False**
Severity	HIGH SEVERITY
Description	In the add and set functions, if unimexIntegration is set to True, then infinite approval is granted via this line: `_lpToken.approve(unimexPool, type(uint256).max);` If, at any point in the future, unimexIntegration is set to False, then the unimexFactory still retains infinite approval over the tokens. If it is malicious, it would then be able to steal funds from the Masterchef at any point in the future.
Recommendation	Consider setting approval to 0 in the set function if unimexIntegration is set to False (after previously being set to True): `lpToken.approve(address(unimexPool), 0);`
Resolution	RESOLVED Approvals are revoked if unimexIntegration is set to False.

Issue #03	**Funds can be stolen if `unimexFactory` is malicious**
Severity	HIGH SEVERITY
Description	Various functions within the contract make external calls to `unimexFactory`. Should the `unimexFactory` address be set to a malicious contract in the constructor, it may be able to steal users' funds. This can be done at any time as the Masterchef owner is able to set any pools to turn on `unimexIntegration`. The steps below detail how funds may be stolen: 1. Set `unimexIntegration` to True. 2. The token balance in the Masterchef contract contract is sent to the `unimexFactory` contract each instance deposit is called. 3. If `unimexFactory` is malicious, funds can be siphoned out there. Additionally, if a pool's `unimexIntegration` is currently `False`, it can still be set to `True` at any point in the future and deposit users' funds into `unimexFactory`. Furthermore, should any of these external calls fail for any reason, it may cause deposits and withdraws to revert, which essentially locks out users' funds.
Recommendation	Consider being transparent with your community and prominently declare and display the `unimexFactory` contract address so that users may be able to verify that it is indeed non-malicious. As the `unimexFactory` contract was not included in this audit scope, we are unable to verify at this stage that it does not contain rug vectors.
Resolution	RESOLVED The client has stated that the external UnimexFactory and UnimexPools contracts have been deployed and operational for over 6 months with no issues, and that they implicitly trust that the contracts are safe. Note that Paladin was not contracted to audit both contracts, and thus cannot comment on their safety.

Issue #04	Contract makes external calls to an unknown contract
Severity	MEDIUM SEVERITY
Description	Various functions within the SpaceX Masterchef contract make external calls to the `unimexFactory`, and as the contract in question was not included in this audit scope, we are unable to verify whether the contract is malicious, nor are we able to gain an understanding of the logic of the functions within. Listed below are 2 examples of functions that make external calls and thus require more extensive scrutiny.

Relevant Functions	**The set function**
	`function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, bool _unimexIntegration) external onlyOwner {`
	`require(_depositFeeBP <= 1000, "set: invalid deposit fee basis points");`
	`totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);`
	`poolInfo[_pid].allocPoint = _allocPoint;`
	`poolInfo[_pid].depositFeeBP = _depositFeeBP;`
	`poolInfo[_pid].unimexIntegration = _unimexIntegration;`
	`IERC20 lpToken = poolInfo[_pid].lpToken;`
	`IUniMexPool unimexPool = IUniMexPool(unimexFactory.getPool(address(lpToken)));`
	`if(_unimexIntegration) {`
	`//deposit lp balance to the pool`
	`require(address(unimexPool) != address(0), "set: no unimex pool");`
	`lpToken.approve(address(unimexPool), type(uint256).max);`
	
	`unimexPool.deposit(lpToken.balanceOf(address(this)));`
	`} else {`
	`uint256 depositedBalance = unimexPool.correctedBalanceOf(address(this));`
	`if(depositedBalance > 0) {`
	`unimexPool.withdraw(depositedBalance);`
	`}`
	`}`

The claimUnimexRewards function

```
function claimUnimexRewards(address lpToken) private
returns (uint256) {
    IUniMexPool unimexPool =
IUniMexPool(unimexFactory.getPool(lpToken));
    if(unimexPool.dividendsOf(address(this)) > 0) {
        uint256 wethBalanceBefore =
WETH.balanceOf(address(this));
        unimexPool.claim();
        return
WETH.balanceOf(address(this)).sub(wethBalanceBefore);
    } else {
        return 0;
    }
}
```

Recommendation

As these functions make external calls to a contract outside this audit scope, we are unable to verify the safety and integrity of the contrat in question. This can be marked as Resolved if the client acknowledges this issue, and commits to ensuring that the `unimexFactory` contract is non-malicious, extensively tested, and does not contain hard rug risks.

Resolution

RESOLVED

The client has stated that the external UnimexFactory and UnimexPools contracts have been deployed and operational for over 6 months with no issues, and that they implicitly trust that the contracts are safe. Note that Paladin was not contracted to audit both contracts, and thus cannot comment on their safety.

Issue #05	**Transfers to the `unimexFactory` may not account for tokens with transfer taxes**
Severity	MEDIUM SEVERITY
Description	The SpaceX Masterchef contract correctly accounts for transfer tax tokens using Uniswap's before-after method. However, as tokens are transferred to and from the `unimexFactory` contract, we are unsure whether that external contract accurately accounts for these deflationary tokens. Additionally, each transfer incurs transfer taxes, which reduces user balances. Governance also has the privilege to abuse `unimexIntegration`, arbitrarily setting it to `True` and `False` repeatedly to cause users' funds to be drained via the transfer tax mechanism.
Recommendation	As the `unimexFactory` contract was not included in this audit scope, we are unable to confirm whether the external contract correctly accounts for transfer tax tokens. A short-term solution to prevent the aforementioned governance abuse is to set the Masterchef contract behind a sufficiently long Timelock. A long-term solution would be to fundamentally revamp the functions to prevent this from occurring, such as removing the ability to set `unimexIntegration` to `False` once it has been set to `True`, though we understand there may be instances when the protocol no longer wishes to support `unimexIntegration` for a pool and thus require this privilege.
Resolution	RESOLVED The client has stated that the UnimexFactory contract does not support tokens with a transfer tax, and that they will take absolute care to ensure that the Masterchef contract does not add any deflationary tokens as pools.

Issue #06	Contract balance may be miscalculated if WETH is deposited to `unimexFactory`
Severity	MEDIUM SEVERITY
Description	If WETH is added as a pool in the Masterchef and subsequently `unimexIntegration` is set to `True` and thus deposited into `unimexFactory`, then it may result in all WETH being deposited in that external contract. There would therefore be no WETH to be paid out as dividends. Additionally, as all WETH is deposited into `unimexFactory`, it may be the case that all WETH could then potentially be claimed as rewards by stakers in the calculation done in claimUnimexRewards.
Recommendation	The most straightforward solution would be to ensure that WETH is either not supported as a pool, or `unimexIntegration` is not set to True if WETH is to be added as a pool.
Resolution	RESOLVED The client has stated that they will not add in a WETH pool to the Masterchef.

Issue #07	**The pendingSpaceX function will revert if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	In the `pendingSpaceX` function, 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 `lastRewardBlock` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.number` and `lpSupply`, like so: `if (block.number > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	RESOLVED

Issue #08	Deposit fees may accrue as dust in the Masterchef
Severity	LOW SEVERITY

Location	Lines 263-265
	`uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);`
	`pool.lpToken.safeTransfer(feeAddress, depositFee.div(2));`
	`pool.lpToken.safeTransfer(vaultAddress, depositFee.div(2));`

Description	As there may be cases when `depositFee` is rounded down due to Solidity's lack of support for floating point numbers, this will result in instances when the sum of deposit fees received by `feeAddress` and `sharesAddress` do not accurately sum to `depositFee`.

Recommendation	Consider instead to transfer the residual deposit fees to `vaultAddress`, as such:
	`uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);`
	`uint256 feeAddressAmount = depositFee.div(2);`
	`pool.lpToken.safeTransfer(feeAddress, feeAddressAmount);`
	`pool.lpToken.safeTransfer(vaultAddress, depositFee.sub(feeAddressAmount));`

Resolution	RESOLVED

Issue #09	**spaceXToken and WETH can be made immutable**
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers.
Recommendation	Consider making the above variables explicitly immutable.
Resolution	RESOLVED

Issue #10	**`unimexFactory` can be hard-coded rather than set in constructor**
Severity	INFORMATIONAL
Description	Various functions within the contract make external calls to `unimexFactory`. Should the `unimexFactory` address be set incorrectly, then functions such as `set`, `deposit`, `withdraw`, `safeWethTransfer` and `claimUnimexRewards` may not work as intended.
Recommendation	Consider hard-coding the `unimexFactory` contract address rather than initializing it in the constructor, as this prevents any unintended errors from occurring.
Resolution	RESOLVED The client has stated that they would ensure that the UnimexFactory address is initialized correctly.

Issue #11	Pools use the contract balance to figure out the total deposits
Severity	INFORMATIONAL
Description	As with pretty much all Masterchefs, the total number of tokens in the Masterchef contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the Masterchef. More severely, because the native token is constantly minted, this will cause severe dilution on the native token pool.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits. Each `lpToken.balanceOf(address(this))` query can then be replaced with this `lpSupply` as well.
Resolution	ACKNOWLEDGED

Issue #12	There are no sanity checks in the `setReferralAddress` function
Severity	INFORMATIONAL
Description	A lot of functionality can break if the referral address is updated to a value that is not a referral contract. Furthermore, the referral contract could be upgraded by governance to a malicious one that sets themselves as the referral of everyone – this would allow them to mint up to 10% of all emissions to their own address.
Recommendation	Consider making the referral address non-upgradeable (only settable once) to ensure that functionality can never break, such as setting it in the constructor. We rarely ever see a project updating their referral after it is initially set.
Resolution	RESOLVED

Issue #13	**safeWethTransfer lacks non-zero safeguard**
Severity	INFORMATIONAL
Description	There may be instances in the deposit and withdraw functions that `safeWethTransfer` attempts to send 0 tokens.

Recommendation	Consider adding in a non-zero if statement before each `safeWethTransfer`:
	`uint256 wethDivs = pendingWeth(_pid, msg.sender);`
	`if (wethDivs > 0) {`
	`  safeWethTransfer(msg.sender, wethDivs);`
	`}`

Resolution	RESOLVED

Issue #14	**Lack of events for updateStartBlock, setReferralCommissionRate, set and add**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY