PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Swift Finance

22 October 2021

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 3

1 Overview 4

1.1 Summary 4

1.2 Contracts Assessed 4

1.3 Findings Summary 5

1.3.1 MasterChef 6

1.3.2 SwiftStakingPool 7

2 Findings 8

2.1 MasterChef 8

2.1.1 Privileged Roles 8

2.1.2 Issues & Recommendations 9

2.2 SwiftStakingPool 19

2.2.1 Privileged Roles 19

2.2.2 Issues & Recommendations 20

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 Swift Finance on the Avalanche 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	Swift Finance
URL	https://swiftfinance.farm/
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`MasterChef`	`0xF0eE9f1ffB32c6CeeF7e491DaA189Cc74352b8DB`	MATCH
`SwiftStakingPool`	`0xB4569DdF3dEAb09CC94e5c6697E001bF7EE70884` (USDC.e) `0x9c88dfaEb52B126bE5ED3d554f82BD0385953085` (wAVAX) `0x1D14AF50A6691E2aF983Dd32D715c674ca8Ab05c` (wETH)	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	1	-	1
Medium	3	3	-	-
Low	7	-	-	7
Informational	14	-	-	14
Total	**26**	**4**	**-**	**22**

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 MasterChef

ID	Severity	Summary	Status
01	HIGH	Severely excessive rewards issue when a token with a transfer tax is added	ACKNOWLEDGED
02	HIGH	Token ownership can be transferred to mint tokens and dump	RESOLVED
03	MEDIUM	Inaccurate staking rewards being given to users	RESOLVED
04	MEDIUM	Setting `devAddress` or `feeAddress` to the zero address will break most functionality	RESOLVED
05	LOW	`updateEmissionRate` has no maximum safeguard	ACKNOWLEDGED
06	LOW	Adding EOA or non-token contract as a pool will break `updatePool` and `massUpdatePools`	ACKNOWLEDGED
07	LOW	The `pendingSwift` function will revert if `totalAllocPoint` is zero	ACKNOWLEDGED
08	INFO	`swiftToken` can be made immutable	ACKNOWLEDGED
09	INFO	Pools use the contract balance to figure out the total deposits	ACKNOWLEDGED
10	INFO	`rewardsUpdateFrequency` can be removed	ACKNOWLEDGED
11	INFO	Minting can slightly exceed maximum token supply	ACKNOWLEDGED
12	INFO	There are no sanity checks on the `setReferralAddress` function	ACKNOWLEDGED
13	INFO	Lack of sanity checks in `switchActivePool`	ACKNOWLEDGED
14	INFO	`deposit`, `withdraw` and `emergencyWithdraw` can be made `external`	ACKNOWLEDGED
15	INFO	Lack of events for `add`, `set`, `setReferralCommissionRate`, `updateStartTimestamp`, `forceWithdraw` and `transferSwiftTokenOwnership`	ACKNOWLEDGED

1.3.2 SwiftStakingPool

ID	Severity	Summary	Status
16	MEDIUM	Functions are not secure from reentrancy	RESOLVED
17	LOW	Lack of constructor safeguards	ACKNOWLEDGED
18	LOW	`depositRewards` should be restricted to `onlyOwner`	ACKNOWLEDGED
19	LOW	`setRewardPerBlock` has no maximum safeguard	ACKNOWLEDGED
20	LOW	The `pendingReward` function will revert if `totalAllocPoint` is zero	ACKNOWLEDGED
21	INFO	`stakeToken` and `rewardToken` can be made immutable	ACKNOWLEDGED
22	INFO	`rewardsUpdateFrequency` can be removed	ACKNOWLEDGED
23	INFO	Contract can be refactored as it only contains a single pool	ACKNOWLEDGED
24	INFO	Rewards can be stopped by owner at any time	ACKNOWLEDGED
25	INFO	`deposit`, `withdraw` and `emergencyWithdraw` can be made `external`	ACKNOWLEDGED
26	INFO	Lack of events for `setRewardPerBlock`, `setRewardEndTimestamp` and `updateStartTimestamp`	ACKNOWLEDGED

2 Findings

2.1 MasterChef

The Masterchef is a fork of Goose Finance's Masterchef. A notable feature of forking the latter is the removal of the `migrator` function from Pancakeswap, which of late has been used maliciously to steal user's tokens. We commend Swift Finance on their decision to fork a relatively safer version of the Masterchef.

Deposit fees are capped at 4%, which would be highly appreciated by users knowing that there is an upper limit on fees. Finally, the Masterchef contract uses `block.timestamp` to account for the variable block times in the Avalanche ecosystem, with the emission currently set to 0.05 tokens per second.

2.1.1 Privileged Roles

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

- `add`
- `set`
- `setDevAddress`
- `setFeeAddress`
- `updateEmissionRate`
- `setReferralAddress`
- `setReferralCommissionRate`
- `updateStartTimestamp`
- `switchActivePool`
- `forceWithdraw`
- `transferSwiftTokenOwnership`

2.1.2 Issues & Recommendations

Issue #01	**Severely excessive rewards issue when a token with a transfer tax is added**
Severity	HIGH SEVERITY
Description	When tokens with a transfer tax are added to the pools, this will result in significant excessive rewards. Due to the way the Masterchef handles rewards, rewards can be heavily inflated when the balance of the Masterchef no longer matches that of user deposits. This happens for example with transfer tax tokens. This issue is further amplified on Masterchefs like this one with a referral mechanism, since tokens can be minted directly. This flaw of the Masterchef has recently been exploited on a significant number of projects, all of which their native tokens went to $0 afterwards because the exploit resulted in a large number of native tokens being minted and dumped. This issue was also present in SushiSwap (the original Masterchef). Since they were never meant to have any tokens but LP tokens, it was not a problem there but has become a problem to projects who have started forking it for usage with less standard tokens.
Recommendation	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:

```
uint256 balanceBefore = pool.lpToken.balanceOf(address(this));
pool.lpToken.transferFrom(msg.sender, address(this), _amount);
_amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);
```

Resolution	ACKNOWLEDGED The client has noted this issue and stated they do not plan on adding tokens with a transfer tax.

Issue #02	Token ownership can be transferred to mint tokens and dump
Severity	HIGH SEVERITY
Description	By calling the `transferSwiftTokenOwnership` function, the owner of this contract can transfer token ownership to any address, which would then be able to mint tokens and dump on the market. Note that transferring token ownership would result in `updatePool` failing to `mint` tokens, and thus `deposit` and `withdraw` will revert as well.
Recommendation	Although the comments state that this will only ever be used if the Masterchef is to be upgraded, it nonetheless poses a very big risk to users of the protocol. We highly recommend that this function be removed for ultimate peace of mind. Nonetheless, should this function be desired, then possibly introducing a 3-7 day delay before the function can be executed would give users plenty of time to react accordingly.
Resolution	RESOLVED The function has been removed.

Issue #03	Inaccurate staking rewards being given to users
Severity	MEDIUM SEVERITY

Location	updatePool, lines 234-236
	`uint256 devReward = swiftReward.div(10);`
	`swiftToken.mint(devAddress, devReward);`
	`swiftToken.mint(address(this), swiftReward);`
	
	deposit and withdraw, lines 260-271, 305-312
	`if (user.amount > 0) {`
	`  uint256 pending = user`
	`  .amount`
	`  .mul(pool.accSwiftPerShare)`
	`  .div(1e18)`
	`  .sub(user.rewardDebt);`
	`  if (pending > 0) {`
	`  uint256 devReward = pending.div(10);`
	`  safeSwiftTransfer(msg.sender, pending.sub(devReward));`
	`  payReferralCommission(msg.sender,`
	`pending.sub(devReward));`
	`  }`
	`}`

Description	In the `deposit` and `withdraw` functions, only 90% of pending rewards and referral commissions are given out. However, in the `updatePool` function, 10% of tokens are minted to the dev and 100% to the Masterchef. This would mean that the pending rewards being shown on the website would be inaccurate, as users would only be receiving 90% of them when they harvest. This would mean that 10% of Swift tokens would stay in the Masterchef balance and naturally grow over time. As the Masterchef currently uses the balance of tokens in the contract to calculate rewards, the degree of inaccuracy in reward calculation grows over time as well.

Recommendation	Consider paying out the full amount of pending rewards to users, and minting the full referral commission as well, in both the deposit and withdraw functions, like so:
	
	`safeSwiftTransfer(msg.sender, pending));`
	`payReferralCommission(msg.sender, pending);`

Resolution	RESOLVED The recommendation has been implemented.

Issue #04	**Setting `devAddress` or `feeAddress` to the zero address will break most functionality**
Severity	MEDIUM SEVERITY
Description	Within the token contract, minting or transferring tokens to the zero address will revert the transaction. Deposits and withdrawals will break if the `feeAddress` is ever set to the zero address. Harvesting will fail as well.

Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like:
	`require(devAddress != address(0), "nonzero!");`
	`require(feeAddress != address(0), "nonzero!");`
	to the configuration functions.

Resolution	RESOLVED The requirement has been added.

Issue #05	**`updateEmissionRate` has no maximum safeguard**
Severity	LOW SEVERITY
Description	Projects sometimes accidentally update their emission rate to a severely high number either by accident or with malicious intent.
Recommendation	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_swiftTokenPerBlock <= MAX_EMISSION_RATE,"Too high");`
Resolution	ACKNOWLEDGED

Issue #06	**Adding EOA or non-token contract as a pool will break `updatePool` and `massUpdatePools`**
Severity	LOW SEVERITY
Description	updatePool will always call `balanceOf(address(this))` on the token of this pool.
Recommendation	Consider simply adding a test line in the add function. If the token does not exist, this will make sure the add function fails. `_lpToken.balanceOf(address(this));`
Resolution	ACKNOWLEDGED

Issue #07	**The `pendingSwift` function will revert if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	In the `pendingSwift` 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 `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: `if (block.timestamp > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) { … }`
Resolution	ACKNOWLEDGED

Issue #08	`swiftToken` 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 and saves gas.
Recommendation	Consider making `swiftToken` explicitly `immutable`.
Resolution	ACKNOWLEDGED

Issue #09	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.
Resolution	ACKNOWLEDGED

Issue #10	`rewardsUpdateFrequency` can be removed
Severity	INFORMATIONAL
Description	In the `getMultiplier` function, the reward multiplier is divided by `rewardsUpdateFrequency`, which is 1. Dividing by 1 does not actually achieve anything, and this variable can therefore be removed to reduce the length of the contract.
Recommendation	Consider removing `rewardsUpdateFrequency`.
Resolution	ACKNOWLEDGED

Issue #11	Minting can slightly exceed maximum token supply
Severity	INFORMATIONAL

Description	Currently, the hard-cap of 100,000 tokens is enforced as follows:
	
	Lines 221-226
	`uint256 currentSupply = swiftToken.totalSupply();`
	
	`if (currentSupply >= MAX_SWIFT_SUPPLY) {`
	`  pool.lastRewardTimestamp = block.timestamp;`
	`  return;`
	`}`
	
	However, this only checks that the limit is not reached before the mint. This could mean that for example if the current supply is 99,999 tokens and there is a request to mint 2 tokens, this request will still pass since the supply is not reached and the final supply will be 100,001 tokens.

Recommendation	Consider not minting the tokens in case the supply exceeds the total supply, like so:
	
	`if (swiftToken.totalSupply().add(swiftReward.mul(11).div(10)) <= MAX_SWIFT_SUPPLY) {`
	`  // The whole emission can be mint`
	`  swiftToken.mint(devAddress, swiftReward.dev(10));`
	`  swiftToken.mint(address(this), swiftReward);`
	`} else if (swiftToken.totalSupply() < MAX_SWIFT_SUPPLY) {`
	`  swiftToken.mint(address(this), MAX_SWIFT_SUPPLY.sub(swiftToken.totalSupply()));`
	`}`
	
	This will only ever mint the total supply at most.
	
	Note that this modification should also be made in the referral minting code section.
	
	A shorter but more advanced approach could be to simply wrap all mint statements in try/catch structures. Even if the mint fails, the main transaction will still succeed.

Resolution	ACKNOWLEDGED

Issue #12	There are no sanity checks on the `setReferralAddress` function
Severity	INFORMATIONAL
Description	A lot of functionality can break if the referral address is updated to an address that is not a referral contract.
Recommendation	Consider making the referral address non-upgradeable (only settable once) to ensure that functionality can never break. We rarely ever see a project updating their referral after it is initially set.
Resolution	ACKNOWLEDGED

Issue #13	Lack of sanity checks in `switchActivePool`
Severity	INFORMATIONAL
Description	There is currently no check to ensure that the length of `_newPids` and `_newAllocPoints` matches up, which may result in inaccurate switching of active pools.
Recommendation	Consider adding a check like so: `require(_newPids.length == _newAllocPoints.length);`
Resolution	ACKNOWLEDGED

Issue #14	**`deposit`, `withdraw` and `emergencyWithdraw` can be made `external`**
Severity	INFORMATIONAL
Description	The above functions can be changed from `public` to `external`. 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 making these functions `external`.
Resolution	ACKNOWLEDGED

Issue #15	**Lack of events for `add`, `set`, `setReferralCommissionRate`, `updateStartTimestamp`, `forceWithdraw`, `transferSwiftTokenOwnership`**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	ACKNOWLEDGED

2.2 SwiftStakingPool

The Swift Staking Pool contract allows for users to stake tokens in exchange for `rewardTokens` over time, similar to Synthetix's staking reward contracts. Reward tokens have to be deposited into the contract in order for rewards to be paid out, and are paid out until reward tokens either run out, or the reward period has ended.

2.2.1 Privileged Roles

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

- `setRewardPerBlock`
- `setRewardEndTimestamp`
- `skimStakeTokenFees`
- `updateStartTimestamp`

2.2.2 Issues & Recommendations

Issue #16	Functions are not secure from reentrancy
Severity	MEDIUM SEVERITY
Description	The `deposit`, `withdraw` and `emergencyWithdraw` functions are susceptible to reentrancy, especially if ERC777 tokens are added to the contract. This may result in the contract being inadvertently exploitable, as was the case with the AMP token in Cream Finance just recently.
Recommendation	Consider adding the `nonReentrant` modifier to the `deposit`, `withdraw`, and `emergencyWithdraw` functions, and reordering line items in the functions to ensure best safety practices are adhered to per the Check Effects Interactions pattern.
Resolution	RESOLVED Reentrancy guards have been added to the relevant functions.

Issue #17	Lack of constructor safeguards
Severity	LOW SEVERITY
Description	The constructor currently lacks validation checks. This could lead to the code being deployed with the `stakedToken` address equaling the `rewardToken` address. This specific setup could result in loss of stakes since the stakes could be given out as rewards to users.
Recommendation	Consider validating that the constructor tokens are not equal to each other. `require(_stakeToken != _rewardToken, "same tokens");`
Resolution	ACKNOWLEDGED

Issue #18	`depositRewards` should be restricted to `onlyOwner`
Severity	LOW SEVERITY
Description	Should any user call the `depositRewards` function by accident, or are tricked into doing so, then they will deposit their `rewardTokens` into the Masterchef. This function should only ever be used by the owner to top up `rewardTokens` if needed
Recommendation	Consider restricting this function to only being callable by the owner.
Resolution	ACKNOWLEDGED

Issue #19	`setRewardPerBlock` has no maximum safeguard
Severity	LOW SEVERITY
Description	Projects sometimes accidentally update their emission rate to a severely high number either by accident or with malicious intent. Additionally, this function should be named `setRewardPerSscond`.
Recommendation	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_rewardPerTimestamp <= MAX_EMISSION_RATE,"Too high");` Additionally, consider renaming the function to reflect that the rewards are not given out per block.
Resolution	ACKNOWLEDGED

Issue #20	The `pendingReward` function will revert if `totalAllocPoint` is zero
Severity	LOW SEVERITY
Description	In the `pendingReward` 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 `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: `if (block.timestamp > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) { … }`
Resolution	ACKNOWLEDGED

Issue #21	`stakeToken` and `rewardToken` 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 and saves gas.
Recommendation	Consider making `stakeToken` and `rewardToken` explicitly `immutable`.
Resolution	ACKNOWLEDGED

Issue #22	`rewardsUpdateFrequency` can be removed
Severity	INFORMATIONAL
Description	In the `getMultiplier` function, the reward multiplier is divided by `rewardsUpdateFrequency`, which is 1. Dividing by 1 does not actually achieve anything, and this variable can therefore be removed to reduce the length of the contract.
Recommendation	Consider removing `rewardsUpdateFrequency`.
Resolution	ACKNOWLEDGED

Issue #23	Contract can be refactored as it only contains a single pool
Severity	INFORMATIONAL

Description	Currently, the contract uses arrays in various functions, like so:
	`PoolInfo storage pool = poolInfo[0];`
	`PoolInfo storage pool = poolInfo[_pid];`
	As the contract only uses 1 pool, the contract can be refactored to remove the array and simply just access the `struct` instead. Functions such as `massUpdatePools` can also be removed since there is no need to attempt to call `updatePool` across multiple pools.
	Lines 754, 771
	`mul(pool.allocPoint).div(totalAllocPoint);`
	Finally, this is unnecessary as `pool.allocPoint == totalAllocPoint`.

Recommendation	Consider refactoring the contract to better reflect its single-pool status.
Resolution	ACKNOWLEDGED

Issue #24	Rewards can be stopped by owner at any time
Severity	INFORMATIONAL
Description	The `setRewardEndTimestamp` function can be called to set `rewardEndTimestamp` to any period, and thus potentially end rewards a lot earlier than expected.
Recommendation	This issue can be marked as Resolved by simply setting the contract behind a sufficiently long Timelock.
Resolution	ACKNOWLEDGED

Issue #25	`deposit`, `withdraw` and `emergencyWithdraw` can be made `external`
Severity	INFORMATIONAL
Description	The above functions can be changed from `public` to `external`. 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 making these functions `external`.
Resolution	ACKNOWLEDGED

Issue #26	Lack of events for `setRewardPerBlock`, `setRewardEndTimestamp` and `updateStartTimestamp`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY