PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Preliminary Report

For SafeGains Finance

03 October 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 SafeGains Finance on the Binance 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	SafeGains Finance
URL	https://safegains.finance
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`SafeGains`	`SafeGains.sol`	PENDING

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	4	4	-	-
Low	0	-	-	-
Informational	3	3	-	-
Total	**7**	**7**	**-**	**-**

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 SafeGains

ID	Severity	Summary	Status
01	MEDIUM	Setting `defaultReferrer` or `platfromWallet` to the zero address may break most functionality	RESOLVED
02	MEDIUM	Penalty fees can be bypassed by transferring SBNB, or just making a secondary deposit	RESOLVED
03	MEDIUM	Users can easily circumvent the 45 day penalty period on secondary deposits	RESOLVED
04	MEDIUM	`withdraw` function is not strictly secure from reentrancy	RESOLVED
05	INFO	`ONE_DAY`, `penaltyDays`, `_want`, `REDISTRIBUTION_FEE`, `REFERRAL_FEE` and `PLATFROM_FEE` can be made `constant`	RESOLVED
06	INFO	Typographical errors in variable and function names	RESOLVED
07	INFO	`recordReferral`, `recordReferralCommission`, `setDefaultReferrer`, `setPlatfromWallet`, `deposit`, `withdraw` and `getRefInfo` functions can be made `external`	RESOLVED

2 Findings

2.1 SafeGains

The SafeGains contract allows users to deposit a want token and potentially receive profit in the form of redistribution and penalty fees incurred from other users after 45 days. Conversely, withdrawing earlier than this period may result in the user incurring penalty fees. The current schedule of fees has been specified as 0.8% redistribution fee, 0.2% referral fee, and 0.2% platform fee.

The referral contract is deployed in the constructor of the SafeGains contract, and thus would require manual contract verification. The owner and operator of the referral contract would be set to the SafeGains contract by default.

If no referrer has been specified in the deposit function, then the `defaultReferrer` address (0x91609451B6a5775608787f3Da9501104935D3b25) receives the referral commission. Additionally, all users wishing to withdraw their funds will pay referral commissions to the `defaultReferrer` address rather than their pre-recorded referrer from earlier, if any has been set. Finally, the 0.2% platform fee is paid upon withdrawals to the `platfromWallet` (typographical error by the protocol), which is controlled by the project team.

2.1.1 Token Overview

Address	0x1286BCf476D7C7936Fc6538459f684eF7F989852
Token Supply	Unlimited
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	Up to 1.2%
Pre-mints	None

2.1.2 Privileged Roles

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

- `setDefaultReferrer`
- `setPlatfromWallet`

2.1.3 Issues & Recommendations

Issue #01	**Setting `defaultReferrer` or `platfromWallet` to the zero address may break most functionality**
Severity	MEDIUM SEVERITY
Description	Any attempts to transfer tokens to the zero address will revert the transaction. As such, should the `defaultReferrer` or `platfromWallet` address be set to the zero address, then the `deposit` and `withdraw` functions may fail.

Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like so:
	`require(user != address(0), "non-zero default referrer");`
	`require(user != address(0), "non-zero platform wallet");`
	to the configuration functions.

Resolution	RESOLVED

Issue #02	Penalty fees can be bypassed by transferring SBNB, or just making a secondary deposit
Severity	MEDIUM SEVERITY
Code	

```
function getPenalty(address user) public view returns(uint256[5]
memory){

    uint256[5] memory penalty =
getPenaltyAmount(user,deposits[user],finalBalance(user),depositCh
eckpoint[user]);

    return penalty;
}

function getPenaltyAmount(address _user,uint256
depositAmount,uint256 currentBal, uint256 depositDate) view
public returns(uint256[5] memory){
    uint256 timeDifferenceInDays =
block.timestamp.sub(depositDate).div(ONE_DAY);
    if(timeDifferenceInDays < penaltyDays  || _user ==
lastPersonToDeposit ){
       return [timeDifferenceInDays,0,0,0,0];
    }
    uint256 profit = currentBal.sub(depositAmount);
    uint256 eachDayProfit = profit.div(timeDifferenceInDays);
    uint256 penaltyProfit =
(timeDifferenceInDays.sub(penaltyDays)).mul(eachDayProfit);
    uint256 latePenaltyFee = penaltyProfit.div(2);

    return
[timeDifferenceInDays,latePenaltyFee,profit,penaltyProfit,eachDay
Profit];
}
```

Description	When users first deposit want tokens, they are minted SBNB receipt tokens. These SBNB tokens would then be burned upon withdrawal and redeemed for the deposit amount plus any fees generated. As penalty is calculated based on when and how much the user has deposited, simply sending the SBNB receipt tokens to a secondary address and then calling `withdraw` from there will bypass having to pay any early penalty fees. Additionally, users can simply withdraw with no penalty if they make a secondary deposit and become `lastPersonToDeposit` due to the following line in the `getPenaltyAmount` function: `if(timeDifferenceInDays < penaltyDays  || _user == lastPersonToDeposit ){` When a user has, for example, made a large deposit, they can simply make a very small deposit, and then immediately call `withdraw` to circumvent penalty.
Recommendation	Instead of minting and burning receipt tokens, consider instead to use a mapping that keeps track of user balances. For the secondary issue of being able to withdraw with no penalty if the user is `lastPersonToDeposit`, we are unsure if this is desired behaviour. If not, then consider removing the `_user == lastPersonToDeposit` check. If it is desired, do let us know and we shall mark the issue as Resolved.
Resolution	RESOLVED The client has implemented mapping to keep track of user balances, and the secondary issue is an intended feature.

Issue #03	Users can easily circumvent the 45 day penalty period on secondary deposits
Severity	MEDIUM SEVERITY
Code	

```
function deposit(uint _amount,address referrer) public
nonReentrant {
    uint256 _pool = balance();
    want().safeTransferFrom(msg.sender, address(this), _amount);

    uint256 _after = balance();
    _amount = _after.sub(_pool);
    _amount = chargeFee(msg.sender,_amount,referrer,false);

    uint256 shares = 0;
    if (totalSupply() == 0) {
        shares = _amount;
    } else {
        shares = (_amount.mul(totalSupply())).div(_pool);
    }
    if(deposits[msg.sender] == 0){
      depositCheckpoint[msg.sender] = block.timestamp;
    }
    if(depositCheckpoint[msg.sender].add(penaltyDays.mul(ONE_DAY)
)  > block.timestamp && deposits[msg.sender] <= _amount){
        depositCheckpoint[msg.sender] = block.timestamp;
        emit onPenalyExtends();
    }
    deposits[msg.sender] = deposits[msg.sender].add(_amount);

    secondLastDepositPerson = lastPersonToDeposit;
    lastPersonToDeposit  = msg.sender;
    emit onDeposit(_amount,referrer);

    _mint(msg.sender, shares);
}
```

Description	The highlighted lines above indicate that penalty extension is applied if the secondary deposit's amount is more than the first deposit amount. Should the user instead simply just deposit an amount slightly lower than the first deposit amount, then `deposits[msg.sender]` is incremented and no penalty extension is applied, illustrated below: First deposit `_amount` = 500 tokens Second deposit `_amount` = 450 tokens `deposits[msg.sender] <= _amount` thus returns `false`.
Recommendation	If penalty extension is desired, consider instead to apply the extension period if `_amount` to be deposited is > 0.
Resolution	RESOLVED The client has stated that this is an intended feature.

Issue #04	`withdraw` function is not strictly secure from reentrancy
Severity	MEDIUM SEVERITY
Description	The `withdraw` function is 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 `withdraw` function, and reordering line items in the function to ensure best safety practices are adhered to per the Check Effects Interactions pattern.
Resolution	RESOLVED

Issue #05	`ONE_DAY`, `penaltyDays`, `_want`, `REDISTRIBUTION_FEE`, `REFERRAL_FEE` and `PLATFROM_FEE` can be made `constant`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers.
Recommendation	Consider making the above variables explicitly `constant`.
Resolution	RESOLVED

Issue #06	Typographical errors in variable and function names
Severity	INFORMATIONAL

Description	In the contract, platform has instead been incorrectly spelled as platfrom.
	
	`PLATFROM_FEE`
	`platfromWallet`
	`setPlatfromWallet`
	`platfromFee`

Recommendation	Consider correcting these to platform instead.
Resolution	RESOLVED

Issue #07	`recordReferral`, `recordReferralCommission`, `setDefaultReferrer`, `setPlatfromWallet`, `deposit`, `withdraw` and `getRefInfo` functions 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	RESOLVED


PALADIN
BLOCKCHAIN SECURITY