



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Glide Finance

18 September 2021



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	4
1 Overview	5
1.1 Summary	5
1.2 Contracts Assessed	6
1.3 Findings Summary	7
1.3.1 GlideToken	8
1.3.2 Sugar	8
1.3.3 MasterChef	9
1.3.4 Router	10
1.3.5 ERC20	10
1.3.6 Pair	10
1.3.7 Factory	10
1.3.8 GlideLibrary	11
1.3.9 RouterHelper	11
1.3.10 Helper Libraries	11
1.3.11 Timelock	11
2 Findings	12
2.1 GlideToken	12
2.1.1 Token Overview	13
2.1.2 Privileged Operations	13
2.1.3 Issues & Recommendations	14
2.2 Sugar	17
2.2.1 Token Overview	17
2.2.2 Privileged Operations	17
2.2.3 Issues & Recommendations	18
2.3 MasterChef	20

2.3.1 Privileged Operations	21
2.3.2 Issues & Recommendations	22
2.4 Router	39
2.4.1 Issues & Recommendations	40
2.5 ERC20	41
2.5.1 Issues & Recommendations	42
2.6 Pair	44
2.6.1 Issues & Recommendations	45
2.7 Factory	46
2.7.1 Issues & Recommendations	47
2.8 GlideLibrary	49
2.8.1 Issues & Recommendations	49
2.9 RouterHelper	50
2.9.1 Issues & Recommendations	50
2.10 Helper Libraries	51
2.10.1 Math, SafeMath, TransferHelper, UQ112x112	51
2.10.2 Issues & Resolutions	51
2.11 Timelock	52
2.11.1 Issues & Resolutions	52



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 advocacy 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 the Glide Finance on the Elastos 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	Glide Finance
URL	https://glidefinance.io/
Platform	Elastos
Language	Solidity



1.2 Contracts Assessed

Name	Contract	Live Code Match
GlideToken	GlideToken.sol	PENDING
Sugar	Sugar.sol	PENDING
MasterChef	MasterChef.sol	PENDING
Router	Router.sol	PENDING
ERC20	ERC20.sol	PENDING
Pair	Pair.sol	PENDING
Factory	Factory.sol	PENDING
GlideLibrary	GlideLibrary.sol	PENDING
RouterHelper	RouterHelper.sol	PENDING
Helper Libraries	Math, SafeMath, TransferHelper, UQ112x112	PENDING
Timelock	Timelock.sol	PENDING



1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	3	3	-	-
● Medium	4	3	-	1
● Low	10	8	1	1
● Informational	15	12	-	2
Total	32	26	1	4

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 GlideToken

ID	Severity	Summary	Status
01	LOW	mint uses raw addition	RESOLVED
02	LOW	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	PARTIAL
03	INFO	Governance functionality is broken	RESOLVED
04	INFO	delegateBySig can be frontrun and cause denial of service	RESOLVED

1.3.2 Sugar

ID	Severity	Summary	Status
05	LOW	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	ACKNOWLEDGED
06	INFO	Governance functionality is broken	RESOLVED
07	INFO	delegateBySig can be frontrun and cause denial of service	RESOLVED

1.3.3 MasterChef

ID	Severity	Summary	Status
08	HIGH	Severely excessive rewards issue when a token with a transfer tax is added	RESOLVED
09	HIGH	Syrup bug is present	RESOLVED
10	MEDIUM	Functions are not secure from reentrancy	RESOLVED
11	MEDIUM	Duplicated pools may be added to the Masterchef	RESOLVED
12	MEDIUM	setStartBlock does not adjust pools' lastRewardBlock	RESOLVED
13	MEDIUM	Token ownership can be transferred	ACKNOWLEDGED
14	LOW	updatePool may fail to mint when supply approaches maximum supply	RESOLVED
15	LOW	Setting to the zero address will break deposit and withdraw functions	RESOLVED
16	LOW	Contract uses raw arithmetic and may be susceptible to underflows and overflows	RESOLVED
17	LOW	Adding EOA or non-token contract as a pool is possible	RESOLVED
18	LOW	The pendingGlide function will revert if totalAllocPoint is zero	RESOLVED
19	LOW	reductionPeriod and bonusReductionPeriod lack safeguards	RESOLVED
20	INFO	glide, sugar, treasuryaddr can be made immutable	RESOLVED
21	INFO	Typos in the contract, or confusing naming conventions	RESOLVED
22	INFO	Pools use the contract balance to figure out the total deposits	RESOLVED
23	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED
24	INFO	dev function can be renamed	RESOLVED
25	INFO	Log event can be removed	RESOLVED
26	INFO	msg.sender is already an address and does not have to be cast to address	RESOLVED
27	INFO	getGlideRewardPerBlock should be renamed	RESOLVED

1.3.4 Router

ID	Severity	Summary	Status
28	INFO	Phishing is possible by a malicious frontend by adjusting routes, tokens or from parameters (also present in Uniswap)	ACKNOWLEDGED

1.3.5 ERC20

ID	Severity	Summary	Status
29	LOW	Approval event is not emitted if allowance is changed in transferFrom as suggested in the ERC-20 Token Standard (also present in Uniswap)	RESOLVED
30	INFO	Permit can be frontrun to prevent someone from calling removeLiquidityWithPermit (also present in Uniswap)	ACKNOWLEDGED

1.3.6 Pair

ID	Severity	Summary	Status
31	INFO	Pairs without supply but with a partial reserve might crash the frontend if the user wants to swap on this pair (this issue is present in most frontends)	ACKNOWLEDGED

1.3.7 Factory

ID	Severity	Summary	Status
32	HIGH	Users' funds can be stolen via uncapped LP minting	RESOLVED

1.3.8 GlideLibrary

No issues found.

1.3.9 RouterHelper

No issues found.

1.3.10 Helper Libraries

Math, SafeMath, TransferHelper, UQ112x112

No issues found.

1.3.11 Timelock

No issues found.



2 Findings

2.1 GlideToken

The contract allows for Glide tokens to be minted when the `mint` function is called by Owner, who at the time of deployment would be the deployer although 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	50,000,000 (50 million)
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None
Pre-mints	TBD



2.1.2 Privileged Operations

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

- `mint`







2.1.3 Issues & Recommendations

Issue #01	mint uses raw addition
Severity	 LOW SEVERITY
Location	<u>Line 14</u> <code>require(totalSupply() + _amount <= _maxTotalSupply, "ERC20: minting more then MaxTotalSupply");</code>
Description	Despite the risk of overflows being low in this contract, the use of raw addition may cause overflow issues in certain edge cases as the contract is using Solidity version 0.6.12.
Recommendation	Consider either using Solidity versions 0.8.0 or higher, or implementing SafeMath's add rather than using raw addition.
Resolution	 RESOLVED SafeMath has been implemented.

Issue #02	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef
Severity	<div> <div></div> <div>LOW SEVERITY</div> </div>
Description	<p>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.</p> <p>This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.</p>
Recommendation(s)	Consider being forthright if this mint function has been 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	<div> <div></div> <div>PARTIALLY RESOLVED</div> </div> <p>The client will pre-mint some tokens to bootstrap liquidity. Once this has been done and ownership transferred to the Masterchef, we shall mark this issue as Resolved.</p>



Issue #03 Governance functionality is broken	
Severity	 INFORMATIONAL
Description	<p>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.</p> <p>It should be noted that this issue is present in pretty much every single farm out there including PancakeSwap and even SushiSwap.</p>
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

Issue #04 delegateBySig can be frontrun and cause denial of service	
Severity	 INFORMATIONAL
Description	<p>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.</p> <p>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.</p>
Recommendation	Similar to the broken governance functionality issue, this can just be removed.
Resolution	 RESOLVED

2.2 Sugar

This contract mints Syrup tokens when users stake into the first pool in the Masterchef via `enterStaking`, and burns them when users withdraw their funds via `leaveStaking`.

2.2.1 Token Overview



Address	TBC
Token Supply	TBC
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None
Pre-mints	TBD



2.2.2 Privileged Operations



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

- `mint`
- `burn`

2.2.3 Issues & Recommendations

Issue #05	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef
Severity	 LOW SEVERITY
Description	<p>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.</p> <p>This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.</p>
Recommendation	Consider being forthright if this mint function has been 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

Issue #06 Governance functionality is broken	
Severity	 INFORMATIONAL
Description	<p>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.</p> <p>It should be noted that this issue is present in pretty much every single farm out there including PancakeSwap and even SushiSwap.</p>
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

Issue #07 delegateBySig can be frontrun and cause denial of service	
Severity	 INFORMATIONAL
Description	<p>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.</p> <p>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.</p>
Recommendation	Similar to the broken governance functionality issue, this can just be removed.
Resolution	 RESOLVED

2.3 MasterChef

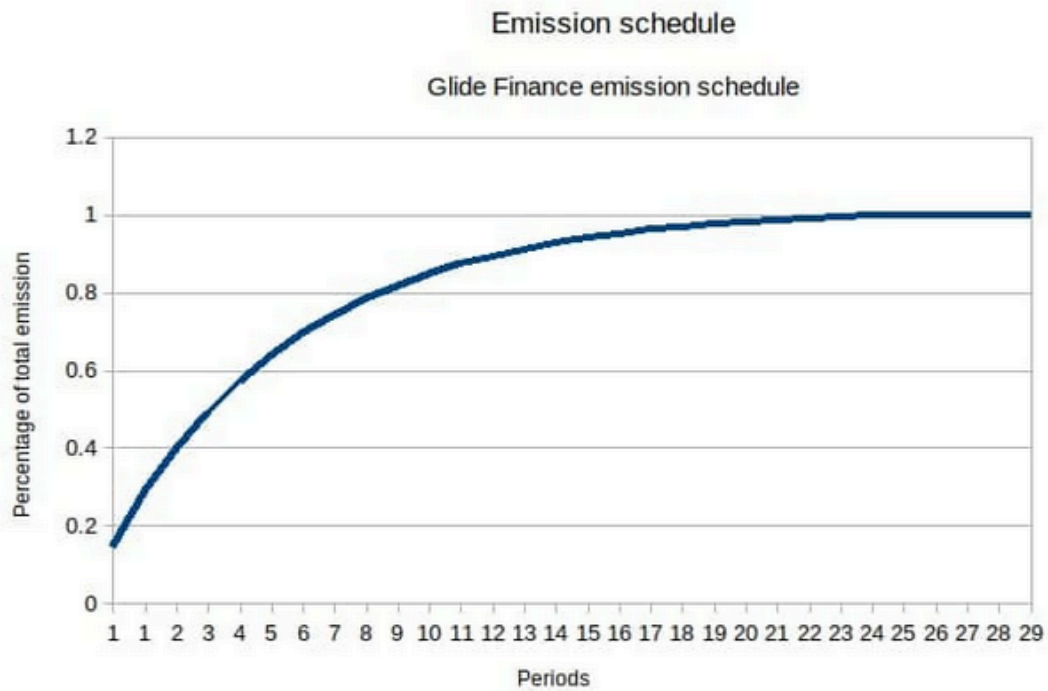
Below is a graphical illustration of the emission schedule for Glide Finance, based on the `glideReward` function. We present this illustration to show that, in its current state, the emission function does look to be functioning as intended, tapering emissions over phases. Note that the minting calculations are highly complex and as such, extensive testing is required should any changes be proposed and executed as it is highly delicate and may pose heightened risks of errors should any parameters be tampered. The rewards mechanism looks to be forked from an MDX vault.

The Glide Finance Masterchef is a fork of Pancakeswap's Masterchef, with the migrator code removed. We commend Glide Finance on their decision to remove the malicious function, which has been used several times in the past to steal users' funds. There are no deposit fees. However, the syrup bug is present ([read more here](#)).

There is a maximum limit of 1000 tokens per block for emissions, though this seems rather high. Emission rewards are distributed according to the following allocation:

- 65% to stakers
- 12.5% to dev
- 22.5% to treasury







2.3.1 Privileged Operations

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

- add
- set
- dev
- setBonusReductionPeriod
- setReductionPeriod
- setStartBlock
- updateEmissionRate
- setGlideTransferOwner
- transferGlideOwnership

2.3.2 Issues & Recommendations

Issue #08	Severely excessive rewards issue when a token with a transfer tax is added
Severity	 HIGH SEVERITY
Description	<p>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.</p> <p>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.</p> <p>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.</p>
Recommendation	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:
	<pre>uint256 balanceBefore = pool.lpToken.balanceOf(address(this)); pool.lpToken.transferFrom(msg.sender, address(this), _amount); _amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);</pre>
Resolution	 RESOLVED
	However, require(_amount > 0, "MasterChef: we dont accept deposits of 0 size"); is redundant.

Issue #09**Syrup bug is present****Severity** HIGH SEVERITY**Description**



The syrup tokens are minted when users staked in the first pool, and burned when the withdraw function is called. Unfortunately, the syrup tokens are not burned when emergencyWithdraw is called, resulting in users being able to exploit the issue and inflate their syrup token balance; In Pancakeswap's case, to the tune of 30 million Syrup tokens.

Recommendation



Consider burning the Syrup tokens in the emergencyWithdraw function, like so:

```
function emergencyWithdraw(uint256 _pid) external
nonReentrant {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    uint256 amount = user.amount;
    user.amount = 0;
    user.rewardDebt = 0;
    if(_pid == 0) {
        syrup.burn(msg.sender, amount);
    }
    pool.lpToken.safeTransfer(address(msg.sender), amount);
    emit EmergencyWithdraw(msg.sender, _pid, amount);
}
```

Resolution RESOLVED

Issue #10	Functions are not secure from reentrancy
Severity	 MEDIUM SEVERITY
Description	All deposit, withdraw and staking-related 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 deposit, withdraw, emergencyWithdraw, enterStaking and leaveStaking functions, and reordering line items in the functions to ensure best safety practices are adhered to per the Check Effects Interactions pattern.
Resolution	 RESOLVED



Issue #11	Duplicated pools may be added to the Masterchef
Severity	 MEDIUM SEVERITY
Description	The add function allows for duplicate pools to be added, which would lead to dilution of emission rewards to stakers.
Recommendation	<p>The addition of a modifier that checks for duplicate pools could help prevent this incident from occurring.</p> <pre>mapping(IBE20 => bool) public poolExistence; modifier nonDuplicated(IBE20 _lpToken) { require(poolExistence[_lpToken] == false, "nonDuplicated: duplicated"); _; }</pre> <p>Alternatively, you could account for this by adding in an lpSupply variable under poolInfo. This has the benefit of accounting for accurately accounting for deposits in the Masterchef.</p>
Resolution	 RESOLVED

Issue #12**setStartBlock does not adjust pools' lastRewardBlock****Severity** MEDIUM SEVERITY**Description**

The setStartBlock function can only be called even after pools have been added. Unfortunately, in its current state, the function does not update each pool's lastRewardBlock, which may result in the emission rewards not starting at the desired block.

Recommendation

Consider implementing the following for setStartBlock that uses a for loop to update each pool (Note that this may fail if there are a sufficiently large number of pools):

```
function setStartBlock(uint256 _newStartBlock) external
onlyOwner {
    require(block.number < startBlock, "cannot change start
block if farm has already started");
    require(block.number < _newStartBlock, "cannot set start
block in the past");
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        PoolInfo storage pool = poolInfo[pid];
        pool.lastRewardBlock = _newStartBlock;
    }
    startBlock = _newStartBlock;
}
```

Note that this loop may run out of gas if a very significant amount of pools are added.

Resolution RESOLVED

Issue #13**Token ownership can be transferred****Severity** MEDIUM SEVERITY**Location**Lines 422-437

```
function setGlideTransferOwner(address _glideTransferOwner) external {
    require(msg.sender == glideTransferOwner);
    glideTransferOwner = _glideTransferOwner;
    emit SetGlideTransferOwner(msg.sender, _glideTransferOwner);
}

/**
 * @dev DUE TO THIS CODE THIS CONTRACT MUST BE BEHIND A TIMELOCK (Ideally 7)
 * THIS FUNCTION EXISTS ONLY IF THERE IS AN ISSUE WITH THIS CONTRACT
 * AND TOKEN MIGRATION MUST HAPPEN
 */
function transferGlideOwnership(address _newOwner) external {
    require(msg.sender == glideTransferOwner);
    glide.transferOwnership(_newOwner);
    emit TransferGlideOwnership(msg.sender, _newOwner);
}
```

Description

At any point in the future, the owner of the Masterchef has the ability to transfer ownership of the Glide token to any address, which allows them to mint an infinite number of tokens and dump them on users. This issue is marked Medium Risk because the client acknowledges the risk of this function in their comments, and resolves to place the Masterchef behind a 7 day Timelock so that users may monitor the queued transactions and react appropriately, though we are still obligated to point out this risk so that users are aware.

Recommendation

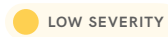
We realize the need to have these functions to transfer ownership of the token contract in cases of emergencies, though we must firstly recommend that these functions be removed, if at all possible, as they pose a significant danger to users. Should the client wish to retain these functions, we will mark the issue as Resolved once the 7-day Timelock has been implemented.

Resolution

ACKNOWLEDGED

The client states that they may wish to upgrade the staking contracts in the future, and wish to retain this function and this function will be set behind a 7 day Timelock. We shall mark the issue as Resolved once this is done.



Issue #14**updatePool may fail to mint when supply approaches maximum supply****Severity****Description**

Given the following scenario:

- Max supply of 50 million tokens
- Glide's total supply is 49,999,999
- glideReward is 5 tokens

The updatePool function will fail to mint the final 1 token, because it fails to pass the if statement on line 280 :

```
if (glide.totalSupply() + glidePerBlockCalculated < glide.getMaxTotalSupply()) {
```

Additionally, there may be precision issues in the mint rewards due to Solidity's lack of support for floating points. Note also that if any tokens are burned at any point after max supply has been reached, then the Masterchef would thus be able to resume token emission minting once again (as supply falls below maxTotalSupply).

Recommendation

Consider adjusting the token minting portion of updatePool to the following, which will ensure that all tokens can be minted, including close to max supply:

```

uint256 glidePerBlockCalculated = getGlideReward(pool.lastRewardBlock);
// add this check so this function does not fail on require on mint. Only mint
// difference between max and totalSupply if it exceeds.
uint256 glideReward =
glidePerBlockCalculated.mul(pool.allocPoint).div(totalAllocPoint);
if (glide.totalSupply().add(glideReward) > glide.getMaxTotalSupply()) {
    glideReward = (glide.getMaxTotalSupply()).sub(glide.totalSupply());
}
if (glideReward != 0) {
    uint256 glideRewardSugar = glideReward.mul(650).div(1000); //65% go to
    sugar
    uint256 glideRewardDev = glideReward.mul(125).div(1000);
    glide.mint(address(sugar), glideRewardSugar);
    glide.mint(devaddr, glideRewardDev ); //12.5% go to dev addr
    glide.mint(treasuryaddr,
glideReward.sub(glideRewardSugar).sub(glideRewardDev)); //22.5% go to treasury
    pool.accGlidePerShare =
pool.accGlidePerShare.add(glideRewardSugar.mul(1e12).div(lpSupply));
    pool.lastRewardBlock = block.number;
}



```

Resolution







RESOLVED






Issue #15	Setting devaddr to the zero address will break deposit and withdraw functions
Severity	 LOW SEVERITY
Description	Any attempt to transfer or mint tokens to the zero address will revert, thus causing deposits and withdrawals to revert if the devaddr is ever set to the zero address.
Recommendation	<p>To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like the following:</p> <pre>require(_devaddr != address(0), "!nonzero");</pre> <p>to the dev function.</p>
Resolution	 RESOLVED



Issue #16	Contract uses raw arithmetic and may be susceptible to underflows and overflows
Severity	 LOW SEVERITY
Location	<p><u>Line 154</u></p> <pre>return glidePerBlock.mul(75).div(100).mul(85 ** (phaseNumber - 2)).div(100 ** (phaseNumber - 2));</pre> <p><u>Line 177</u></p> <pre>uint256 r = (n - 1).mul(reductionPeriod).add(startBlock).add(bonusReductionPeriod);</pre> <p><u>Line 248</u></p> <pre>if (glide.totalSupply() + glidePerBlockCalculated < glide.getMaxTotalSupply()) {</pre> <p><u>Line 280</u></p> <pre>if (glide.totalSupply() + glidePerBlockCalculated < glide.getMaxTotalSupply()) {</pre>
Description	As the contract is using Solidity version 0.6.12, the use of raw addition and subtraction may cause overflow and underflow issues in certain edge cases.
Recommendation	Consider either using Solidity versions 0.8.0 or higher, or implementing SafeMath's add and sub rather than using raw addition and subtraction.
Resolution	 RESOLVED SafeMath has been implemented.

Issue #17	Adding EOA or non-token contract as a pool is possible
Severity	 LOW SEVERITY
Description	updatePool will always call <code>balanceOf(address(this))</code> on the token of this pool in the Masterchef, and will fail to do so if it is not a token contract.
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. <code>_lpToken.balanceOf(address(this));</code>
Resolution	 RESOLVED

Issue #18	The pendingGlide function will revert if totalAllocPoint is zero
Severity	 LOW SEVERITY
Description	In the pendingGlide function, at some point a division is made by the <code>totalAllocPoint</code> 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 <code>lastRewardBlock</code> if the <code>totalAllocPoint</code> variable is greater than zero. This check can simply be added to the existing check that verifies the <code>block.number</code> and <code>lpSupply</code> , like so: <pre>if (block.number > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) {</pre>
Resolution	 RESOLVED

Issue #19**reductionPeriod and bonusReductionPeriod lack safeguards****Severity** LOW SEVERITY**Description**


As both `reductionPeriod` and `bonusReductionPeriod` determine the phases, setting these to longer periods may result in the phases actually falling back to an earlier period (from phase 3 back down to phase 2). Changing these periods arbitrarily may result in unintended and dangerous side effects (note that these changes are applied retroactively), and the client should very carefully consider the implications of doing so.

Recommendation

The simplest solution would be to remove the ability to adjust both `reductionPeriod` and `bonusReductionPeriod`. Alternatively, the client may wish to set reasonable minimum safeguards to ensure that both these reduction periods do not result in unintended side effects, or backtracking in phases.

Resolution RESOLVED

Both functions have been removed.



Issue #20**glide, sugar, treasuryaddr 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 #21	Typos in the contract, or confusing naming conventions
Severity	 INFORMATIONAL
Description	<p><u>Line 66</u></p> <pre>// Bounus reduction period</pre> <p>We believe this should be bonus.</p> <p><u>Line 142</u></p> <pre>function phase() public view returns (uint256) {</pre> <p>This function is named the same as the function in line 124, and may be confusing to readers.</p> <p><u>Line 163</u></p> <pre>function reward() public view returns (uint256) {</pre> <p>This function is named the same as the function in line 158, and may be confusing to readers.</p>
Recommendation	To make the contract more consistent and thus readable for third-party reviewers, consider making the necessary corrections.
Resolution	 RESOLVED
	phase and reward are intended features.



Issue #22 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	 RESOLVED

Issue #23 Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	
Severity	 INFORMATIONAL
Description	<p>Within updatePool, accGlidePerShare is based on the lpSupply variable.</p> <pre>pool.accGlidePerShare = pool.accGlidePerShare.add(glideReward.mul(1e12).div(lpSupply));</pre> <p>However, if this lpSupply becomes a severely large value, precision errors may occur due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals.</p>
Recommendation	Consider increasing precision to 1e18 across the entire contract.
Resolution	 RESOLVED

Issue #24	dev function can be renamed
Severity	 INFORMATIONAL
Description	Reviewers may be confused on what the dev function does.
Recommendation	Consider renaming this to setDevAddress.
Resolution	 RESOLVED

Issue #25	Log event can be removed
Severity	 INFORMATIONAL
Location	<u>Line 88</u> event Log(uint256 message); <u>Line 185</u> //emit Log(blockReward);
Description	Line 185 has been commented out, and the event no longer serves a purpose.
Recommendation	As this has been commented out in line 185, consider removing both lines as they do not seem to be used in the contract.
Resolution	 RESOLVED

Issue #26	msg.sender is already an address and does not have to be cast to address
Severity	 INFORMATIONAL
Description	Throughout the codebase, msg.sender is explicitly cast to an address like in the following code: address(msg.sender). Since msg.sender is already an address, this code is redundant (although it does no harm either).
Recommendation	Consider replacing all occurrences of address(msg.sender) with msg.sender.
Resolution	 RESOLVED

Issue #27	getGlideRewardPerBlock should be renamed
Severity	 INFORMATIONAL
Description	The function does not return glide per block, but rather total glide rewards. This is rather confusing to readers and third-party reviewers.
Recommendation	Consider renaming the function to avoid confusion.
Resolution	 RESOLVED



2.4 Router

The Glide AMM protocol, forked with minimal changes from Uniswap, uses the GlideRouter as entry point for users to exchange tokens. The GlideRouter is responsible for determining the swap rate and allowing for user-interactions to be done with safety checks. More specifically, the GlideRouter allows routers to add liquidity, remove liquidity and swap tokens.



2.4.1 Issues & Recommendations



Issue #28	Phishing is possible by a malicious frontend by adjusting routes, tokens or from parameters (also present in Uniswap)
Severity	<div><div></div> INFORMATIONAL</div>
Description	<p>A malicious (for example hacked) frontend can easily mislead users in approving malicious transactions, even if the router matches the address described in this report.</p> <p>An obvious example of how this can be done is by changing the to parameter which indicates to whom tokens or liquidity has to be sent. Other ways to phish could include using malicious routes or tokens.</p>
Recommendation	Consider carefully protecting the frontend and ideally having an unchangeable IPFS fallback implementation for it.
Resolution	<div><div></div> ACKNOWLEDGED</div> <p>The client endeavours to secure their front-end and may consider implementing an IPFS fallback.</p>

2.5 ERC20

The GlideERC20 is an implementation of the [ERC-20 Token Standard](#). It is a clean copy of the related Uniswap contract.



2.5.1 Issues & Recommendations

Issue #29	Approval event is not emitted if allowance is changed in transferFrom as suggested in the ERC-20 Token Standard (also present in Uniswap)
Severity	 LOW SEVERITY
Location	<u>Lines 74-80</u> <pre>function transferFrom(address from, address to, uint value) external returns (bool) { if (allowance[from][msg.sender] != uint(-1)) { allowance[from][msg.sender] = allowance[from] [msg.sender].sub(value); } _transfer(from, to, value); return true; }</pre>
Description	<p>The ERC-20 standard specifies that an approval event should be emitted when the allowance of a user changes. However, within the ERC20 implementation of both Uniswap and Glide, this is not done.</p> <p>You can read more about this improvement in Pull Request #65 of uniswap-core.</p>
Recommendation	Consider adding emit Approval(from, msg.sender, remaining) in transferFrom when allowance is modified.
Resolution	 RESOLVED transferFrom now emits an event equal to the value transferred.

Issue #30**Permit can be frontrun to prevent someone from calling
removeLiquidityWithPermit (also present in Uniswap)****Severity** INFORMATIONAL**Location**Lines 82-94

```
function permit(address owner, address spender, uint value,
uint deadline, uint8 v, bytes32 r, bytes32 s) external {
    require(deadline >= block.timestamp, 'PolyCat:
EXPIRED');
    bytes32 digest = keccak256(
        abi.encodePacked(
            '\x19\x01',
            DOMAIN_SEPARATOR,
            keccak256(abi.encode(PERMIT_TYPEHASH, owner,
spender, value, nonces[owner]++, deadline))
        )
    );
    address recoveredAddress = ecrecover(digest, v, r, s);
    require(recoveredAddress != address(0) &&
recoveredAddress == owner, 'PolyCat: INVALID_SIGNATURE');
    _approve(owner, spender, value);
}
```

Description


Currently, if permit is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up permit transactions in the mempool and execute them before a contract can.

The implications of this issue is that a bad actor could prevent a user from removing liquidity with a permit through the router. It is a denial of service attack which is present in all AMMs but which we have yet to witness being used since there is no profit from it.

Recommendation

Consider this issue if there are ever complaints by users that their removeLiquidityWithPermit transactions are failing. It could be the case that someone is using this vector against them.

We do not recommend changing this behavior since it would cause a lot of extra work modifying the frontend to account for new permit behavior. This issue is also present in Uniswap after all.

Resolution ACKNOWLEDGED

2.6 Pair



The GlidePair is the core component of the Glide AMM protocol and represents a pair of two tokens. Users can add liquidity in this pair by depositing both tokens in an equally valued proportion for others to swap against. It is a clean fork from Uniswap with the main difference being that the owner of the Factory contract can adjust the proportion of fees to LP holders and the owner.

Additionally, there is the high risk that the owner is able to mint an infinite number of LP tokens to themselves, and thus steal users' funds. Please refer to the Factory contract section for a full description of the issue.

Note that $\frac{5}{6}$ of fees (0.25%) go to the fee address, whilst LP holders only receive $\frac{1}{6}$ of fees (0.05%).



2.6.1 Issues & Recommendations

Issue #31	Pairs without supply but with a partial reserve might crash the frontend if the user wants to swap on this pair (this issue is present in most frontends)
Severity	 INFORMATIONAL
Description	<p>A malicious DoS attack we have witnessed in practice is when a project wants to go live through a presale, people can instantiate the pair while there are no tokens yet. The malicious party will then sent some of the counterparty token to this pair so it has a partial balance (eg. 0.1 BNB and 0 tokens). When <code>sync()</code> is then called, the pairs' reserves are updated to account for this balance.</p> <p>Due to a division by zero exception, many frontends can not properly account for this state and will go through a blank page, preventing the original project from adding liquidity through the frontend.</p>
Recommendation	Consider checking whether this is present in the frontend and adding a division by zero handler.
Resolution	 ACKNOWLEDGED
	The client will confirm frontend behaviour.

2.7 Factory

The GlideFactory is the management contract for the Glide Finance AMM. It keeps track of all GlidePairs and allows users to create new ones. Any GlidePair created through the verified factory is immediately verified as well, since the pair is deployed by the verified factory.



2.7.1 Issues & Recommendations

Issue #32 Users' funds can be stolen via uncapped LP minting

Severity

 HIGH SEVERITY

Location

Lines 90-110 (from the GlidePair contract)

```
function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool
feeOn) {
    address feeTo = IGlideFactory(factory).feeTo();
    feeOn = feeTo != address(0);
    uint _kLast = kLast; // gas savings
    if (feeOn) {
        if (_kLast != 0) {
            uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
            uint rootKLast = Math.sqrt(_kLast);
            if (rootK > rootKLast) {
                uint numerator = totalSupply.mul(rootK.sub(rootKLast));
                //If we want to set 0.25% of 0.3% to go to feeTo, then div arg
will be 5 and mul will be 1
                //If we want to set 0.05% of 0.3% to go to feeTo, then div arg
will be 1 and mul will be 5 - UniswapV2Pair.sol default value https://
github.com/Uniswap/uniswap-v2-core/blob/master/contracts/UniswapV2Pair.sol
                uint denominator = (rootK /
IGlideFactory(factory).feeToRateDivArg()).mul(IGlideFactory(factory).feeToRate
MulArg()).add(rootKLast);
                uint liquidity = numerator / denominator;
                if (liquidity > 0) _mint(feeTo, liquidity);
            }
        }
    } else if (_kLast != 0) {
        kLast = 0;
    }
}
```

Lines 56-67 (in this Factory contract)

```
function setFeeToSetter(address _feeToSetter) external override {
    require(msg.sender == feeToSetter, 'Glide: FORBIDDEN');
    feeToSetter = _feeToSetter;
    emit SetAdminAddress(msg.sender, _feeToSetter);
}
```

```
function setFeeToRate(uint256 _rateDivArg, uint256 _rateMulArg) external
override {
    require(msg.sender == feeToSetter, 'Glide: FORBIDDEN');
    require(_rateDivArg > 0, "Glide: FEE_TO_RATE_DIV_OVERFLOW");
    feeToRateDivArg = _rateDivArg;
    feeToRateMulArg = _rateMulArg;
}
```

Description As the dev can set both `feeToRateDivArg` and `feeToRateMulArg`, the denominator in the `_mintFee` function in the `GlidePair` contract can be any number, including a number smaller than the numerator. If `denominator < numerator`, then these lines in the `GlidePair` contract:

```
uint liquidity = numerator / denominator;
if (liquidity > 0) _mint(feeTo, liquidity);
```

would be able to mint LP tokens beyond the 0.3% fee. As it is uncapped, they could, for example, set it such that any amount of LP tokens are minted to the `feeTo` address, which would then be able to withdraw users' funds from the AMM.

Recommendation The most simple solution would be to revert back to the default Uniswap V2 implementation for the `UniswapPair` contract, and remove both `setFeeToSetter` and `setFeeToRate` functions. It is rare that any AMM should have the ability to change the proportion of fees allocated to LP holders and the owner, and will very likely be frowned upon by third-party reviewers including RugDoc. Alternatively, should the client wish to retain these functions, then consider adding safeguards to ensure that the denominator is equal to or greater than the numerator. This ensures that, at the either extreme, the owner will either take the entire 0.3% fee, or none of it.

Resolution



`setFeeToRate` has been removed, and 5% of the LP is minted to the fee address.

2.8 GlideLibrary

The GlideLibrary contract is a dependency contract used to calculate the appropriate trading rates. It is used by the GlideRouter to calculate how many tokens should be sent to the pairs and is thus an important component of the user-facing aspect of the system.

2.8.1 Issues & Recommendations

No issues found.



2.9 RouterHelper

This contract is a helper library to swap tokens, add and remove liquidity. We are unsure where this will be used, and the client should consider clarifying if this will be used in any contracts. Should this be integrated into a token contract, for example, issues may arise as swap and LP tokens will be sent to the contract address rather than the message sender (lines 40, 64 and 85).

2.9.1 Issues & Recommendations

No issues found.



2.10 Helper Libraries

2.10.1 Math, SafeMath, TransferHelper, UQ112x112

Math, SafeMath, TransferHelper and UQ112x112 are various helper libraries which are each identical to the PancakeSwap implementation.

2.10.2 Issues & Resolutions

No issues found.



2.11 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	TBC	The delay indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	3 hours	<p>The <code>minDelay</code> indicates the lowest value that the delay can minimally be set.</p> <p>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 <code>delay</code> can never be lower than that of the <code>minDelay</code> 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.</p>
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.

2.11.1 Issues & Resolutions

No issues found.



PALADIN
BLOCKCHAIN SECURITY