**APWINE MASTERCHEF SMART CONTRACT AUDIT**

by [Manmeet Singh](https://www.linkedin.com/in/manmeet-singh-9733a511b/)

Document version: v1

Stage: First Draft, Delivered for initial review

# Table of contents

[**Table of contents**](#_lwd1a68qt0x4) **2**

[**Introduction**](#_1t1nxn8uh58r) **3**

[Auditing Approach and Methodologies applied](#_aptz2kfpx4bk) 3

[Audit Details](#_g5diui90ixw9) 3

[**Audit Goals**](#_e7voczw1gi7z) **4**

[Security](#_g0t0yd7iagbw) 4

[Sound Architecture](#_ibh1n6pyyv7w) 4

[Code Correctness and Quality](#_el3jtwfjq4yd) 4

[**Issue Categories**](#_7dok2pw6yez0) **5**

[High level severity issues](#_pjv57re7upf6) 5

[Medium level severity issues](#_l66nk7mlkhj1) 5

[Low level severity issues](#_oqelfnz41i6) 5

[**Manual Audit**](#_3nem8i56otkf) **6**

[Low level severity issues](#_o82bumzbrc4m) 6

[Medium level severity issues](#_1zhj0ogv5q92) 7

[Recommendations](#_zcjwxbegkfbu) 8

[**Automated Audit**](#_exqxdgycb99) **9**

[Remix Compiler Warnings](#_2tozq1flkm30) 9

[Contract Library](#_nh1dgmz10qfx) 9

[SmartCheck](#_f96r4mtb57f) 10

[Slither](#_4ggmoe81t3ev) 11

[**Disclaimer**](#_oxzmve6hwbug) **20**

[**Summary**](#_duza2olyg7z5) **20**

# 

# 

# Introduction

This Audit Report mainly focuses on the overall security of APWine Masterchef Smart Contract. With this report, I have tried to ensure the reliability and correctness of the smart contracts involved by a complete and rigorous assessment of their system's architecture and the smart contract codebase.

## Auditing Approach and Methodologies applied

Rigorous testing of the project was performed starting with analyzing the code design patterns in which the smart contract architecture was reviewed to ensure it is structured and had safe usage of third-party smart contracts and libraries.

Then a formal line-by-line inspection of the Smart Contract was performed to find any potential issues like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In Automated Testing, the Smart Contracts were tested using popular automation tools to identify vulnerabilities and security flaws.

The code analysis methodologies included -

* Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
* Analyzing the complexity of the code in-depth and detailed, manual review of the code, line-by-line.
* Deploying the code on testnet using multiple clients to run live tests.
* Analyzing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
* Checking whether all the libraries used in the code are on the latest version.
* Analyzing the security of the on-chain data.

## Audit Details

Project Name: APWine MasterChef

Codebase: <https://github.com/APWine/apwine-amm/blob/main/contracts/MasterChef.sol>

Commit hash: 070a805177655d779e1bb6004df0629c39803b33

Languages: Solidity (Smart contract)

Platforms and Tools: Remix IDE, Hardhat, Solhint, Slither, SmartCheck

# Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient, and working according to the specifications. The audit activities can be grouped into the following three categories:

## Security

Identifying security-related issues within each contract and the system of contract.

## Sound Architecture

Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

## Code Correctness and Quality

A full review of the contract source code. The primary areas of focus include:

* Accuracy
* Readability
* Sections of code with high complexity
* Quantity and quality of test coverage

# 

# 

# 

# Issue Categories

Every issue in this report was assigned a severity level from the following:

## High level severity issues

Issues on this level are critical to the smart contract’s performance/functionality and should be fixed before moving to a live environment.

## Medium level severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

## Low level severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

|  | LOW | MEDIUM | HIGH | RECOMMENDATIONS |
| --- | --- | --- | --- | --- |
| OPEN | 12 | 3 | 3 | 0 |
| CLOSED | 0 | 0 | 0 | 0 |

# 

# 

# 

# Manual Audit

For this section, the code was tested/read line by line. Remix IDE’s JavaScript VM and Rinkeby networks were also used to test the contract functionality.

## Low level severity issues

* ***Status: Open  
    
  Description:****MasterChef.sol, line 3, Rewarder.sol line 3:*The pragma versions used within the contracts are not locked. This can be confusing for a developer and at the same time can create compile issues for other versions.

***Recommendation:***Consider locking the latest version 0.7.6 for deploying the contracts. Solidity source files indicate the versions of the compiler they can be compiled with.

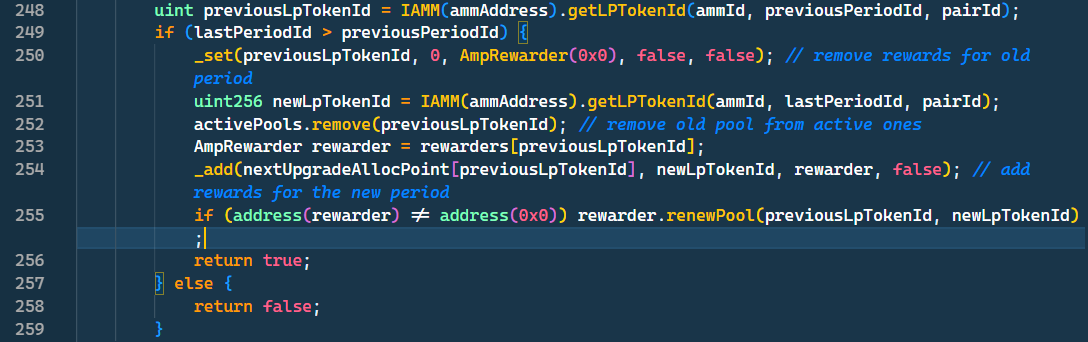
* ***Status: Open  
    
  Description:***The following functions are declared public but are not being called from anywhere else in the contract:  
  *MasterChef.sol:* - Initialize *Rewarder.sol:* - Initialize  
   - setRewardPerSecond  
   - renewPool  
   - setNextRewardPerSecond  
    
  ***Recommendation:***The functions should be declared external. This saves gas on function call and contract deployment.
* ***Status: Open  
    
  Description:****MasterChef.sol, line 63, 65, 67  
  Rewarder.sol, line 42, 43:*In MasterChef.sol, the storage variables lpIDToFutureAddress, nextUpgradeAllocPoint and rewarders use up extra storage space which can be avoided. Also, extra gas is used when setting these variables corresponding to a lpTokenId in the \_add function which can be optimised. Similarly in Rewarder.sol, the storage variables rewardPerSecond and nextRewardPerSecond use up extra storage space which can be avoided.

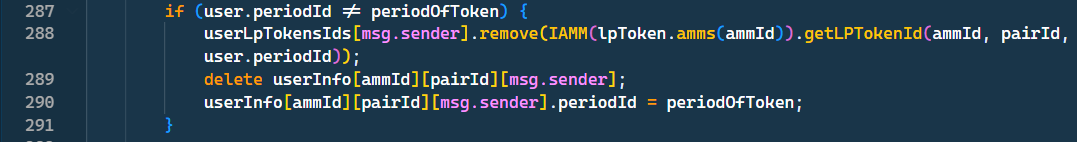
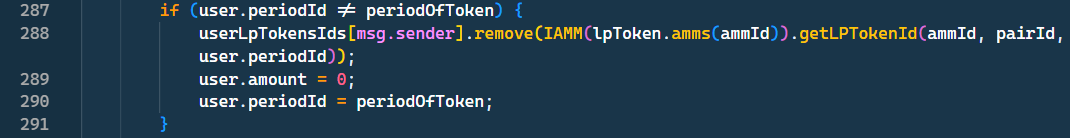
***Recommendation:***The storage variables lpIDToFutureAddress, nextUpgradeAllocPoint and rewarders can be moved to poolInfo struct where the values can be set in a single line of code as all the members of the struct are associated to a lpTokenId. Similarly, the storage variables rewardPerSecond and nextRewardPerSecond can be moved to the poolInfo struct where all the members of the struct will be associated to a pid.

* ***Status: Open  
    
  Description:****MasterChef.sol, line 145-146  
  *The require checks should be done before any storage is modified if possible. It is doable for the above scenario.

***Recommendation:***The above line 145 and 146 can be moved before line 143 after which the rewarder and the future addresses should be stored. This ensures all require checks are done before any storage is modified and saves some additional gas in case this require statement fails.

* ***Status: Open***  
  ***Description:****MasterChef.sol, line 243, 245, 269, 270*In the deposit function, the external calls lpToken.getAMMId(\_lpTokenId) and lpToken.getPairId(\_lpTokenId) are made twice. Once in the \_upgradePoolRewardsIfNeeded function in lines 243, 245 and then in line 269, 270 again. This is redundant and can be optimised to use less gas.   
    
  ***Recommendation:***    
  The external calls lpToken.getAMMId(\_lpTokenId) and lpToken.getPairId(\_lpTokenId) can be made once in the beginning of the deposit function and the values can be passed as parameters to the \_upgradePoolRewardsIfNeeded function.
* ***Status: Open***  
  ***Description:****MasterChef.sol, line 248:*

**In line 248, the getter call for getting previousLpTokenId will use up unnecessary gas in case the if condition inside which the value is used at line 250 is not run. So in case the deposit function is invoked and the lastPeriodId <= previousPeriodId, the getter call in line 248 will run anyway and the returned value will not be used.  
  
***Recommendation:*** The getter call for getting previousLpTokenId should be moved inside the if condition in line 249. This will save gas in case the deposit function is invoked and lastPeriodId <= previousPeriodId.

* ***Status: Open***  
  ***Description:****MasterChef.sol, line 287-291:  
  *Lines 287 to 291 can be optimised to use lesser gas. In 289, deleting the mapping uses up more gas which can be optimised. Also for setting the periodId, the already existing storage variable user can be used to optimize the logic.  
    
  ***Recommendation:***Lines 287 to 291 can be modified to look like:  
  
* ***Status: Open (Subject to discussion)***  
    
  ***Description:****MasterChef.sol, line 246, 267:*In the deposit function, the approach to calculate previousPeriodId/lastPeriodId which is the current period as returned by the FutureVault’s getCurrentPeriodIndex function is different in line 246 and line 267. The difference is in the way FutureVault’s address is fetched to instantiate IFutureVault. In line 246, the function \_upgradePoolRewardsIfNeeded gets the ammId, fetches the ammAddress from the ammId, gets the futureAddress associated to fetched ammAddress and instantiates the IFutureVault to get the previousLpTokenId. Whereas in line 267, the storage variable lpIDToFutureAddress is used to fetch the associated FutureVault’s address and calculate the lastPeriodId. The value for lpIDToFutureAddress was set when the pool was added in a similar way the \_upgradePoolRewardsIfNeeded fetches the associated FutureVault’s address described above.  
    
  ***Recommendation:***It is better to maintain consistency in the way the associated FutureVault’s address is fetched in the same function. The FutureVault’s address can be fetched once in the beginning of the deposit function and then can be passed as a parameter to \_upgradePoolRewardsIfNeeded. The storage variable lpIDToFutureAddress can also be removed if the futureAddress is fetched the way it is done in line 246.
* ***Status: Open***  
    
  ***Description:****MasterChef.sol, line 285, 293, 315, 323, 332, 346, 358  
  Rewarder.sol, line 74:*The transfer amount parameter should be checked to not be 0 before invoking safeTransfer or safeTransferFrom. Invoking an ERC20 token transfer function with a 0 transfer amount parameter uses up unnecessary gas or could cause undesired reverts. The check can be avoided if the value can never be 0.  
    
  ***Recommendation:***The following parameters should be checked to not be 0 using an if condition and invoke the transfer function only if the given parameter is not 0.*MasterChef.sol*: - *Line 285*: pending  
   - *Line 293*: \_amount  
   - *Line 315*: pending  
   - *Line 323*: \_amount  
   - *Line 332*: user.amount  
   - *Line 346*: transferAmount  
   - *Line 358*: \_amountparameter  
    
  *Rewarder.sol*  
   - *Line 74*: pending
* ***Status: Open***  
    
  ***Description:****Rewarder.sol, line 81-91*:  
  The function pendingTokens is not necessary if the return values are arrays whose length will always be 1. The function PendingToken and the public storage variable rewardToken serve the same purpose.  
    
  ***Recommendation:***The function can be removed which can save minor deployment costs.
* ***Status: Open***  
    
  ***Description:****Rewarder.sol, line 130, 143*:   
  Using block timestamps for comparison is dangerous. block.timestamp can be manipulated by miners. This should be avoided wherever possible.  
    
  ***Recommendation:***  
  The implementation can be replaced by block.number and rewards can be calculated on the basis of block difference. This would mean changing variables lastRewardTime to lastRewardBlock, rewardPerSecond to rewardPerBlock and nextRewardPerSecond to nextRewardPerBlock. The logic would then be changed accordingly similar to the implementation in *MasterChef.sol*. Converting the rewards that are in *terms of seconds* to in *terms of blocks* can be estimated if a certain block time is assumed (Recommended is between 10 to 20 seconds).
* ***Status: Open***  
    
  ***Description:****MasterChef.sol, line 263, 307, 328*Unnecessary non reentrant modifiers. Reentrancy protection only needs to be applied if the function is interacting with some 3rd party smart contracts that are not a part of the protocol or not in the protocol’s control. Removing the ReentrancyGuardUpgradeable contract will save deployment costs and removing nonReentrant modifiers will save function call costs.  
    
  ***Recommendation:***  
  The ReentrancyGuardUpgradeable contract and the nonReentrant modifiers can be removed.

## Medium level severity issues

* ***Status: Open***  
    
  ***Description:****MasterChef.sol, line 242-260:*In\_upgradePoolRewardsIfNeeded if lastPeriodId > previousPeriodId and the nextUpgradeAllocPoint is not set for the previousLpTokenId, the allocation point for the newLpTokenId will become 0 which can be undesirable. As setting the nextUpgradeAllocPoint for a lpTokenId is done in another function, it adds a layer of manual intervention for the owner to set the nextUpgradeAllocPoint separately everytime a new pool is added. This can be easily missed by the owner.  
    
  ***Recommendation:***There are 3 possible ways to handle this:  
  - One way to deal with this is to set the nextUpgradeAllocPoint when a pool is added for a lpTokenId in the \_add function by taking another parameter. This removes a need for calling an extra function saving fees.  
  - Another way is to maintain a global default storage value like defaultNextUpgradeAllocPoint which can be also updated by the owner if needed. This value will be set by default for a pool’s nextUpgradeAllocPoint when a new pool is added.  
  - Another less elegant way would be to check in \_upgradePoolRewardsIfNeeded that if the nextUpgradeAllocPoint for a pool is 0, then use the allocPoint of the previousLpTokenId. This approach may not be ideal as the nextUpgradeAllocPoint might be desired to be 0.
* ***Status: Open***  
    
  ***Description:****MasterChef.sol, line 328-340:*The function emergencyWithdraw does not remove \_lpTokenId from the userLpTokensIds enumerable set upon withdrawal of lp tokens. This can create confusion when getUserLpTokenIdList is called after emergencyWIthdraw.  
    
  ***Recommendation:***It is recommended to add userLpTokensIds[msg.sender].remove(\_lpTokenId) after line 334 in the function emergencyWithdraw.
* ***Status: Open***  
    
  ***Description:****Rewarder.sol, line 63-79, 100-104, 106-109, 111-114:*rewardPerSecond and nextRewardPerSecond have no default values. This could lead to issues when a new pool is created in the function onAPWReward(which calls the updatePool function) and the owner has not set the values for rewardPerSecond and nextRewardPerSecond which will be 0 by default. Also if nextRewardPerSecond is set as 0 by default, the renewPool function could set the rewardPerSecond for the new pid to 0 which may be undesirable. By adding default values, a layer of manual intervention can be reduced from the owner perspective where they will not have to call the setRewardPerSecond and setNextRewardPerSecond everytime a new poolId entry is created and will only call it if there is a need for an update in those values.  
    
  ***Recommendation:***The value for mappings rewardPerSecond and nextRewardPerSecond can be set in the updatePool function using the default storage variables or by passing 2 more parameters. The default values can be initialised when the contract is deployed and an additional onlyOwner update function can be created if the default values need any updates.

## High level severity issues

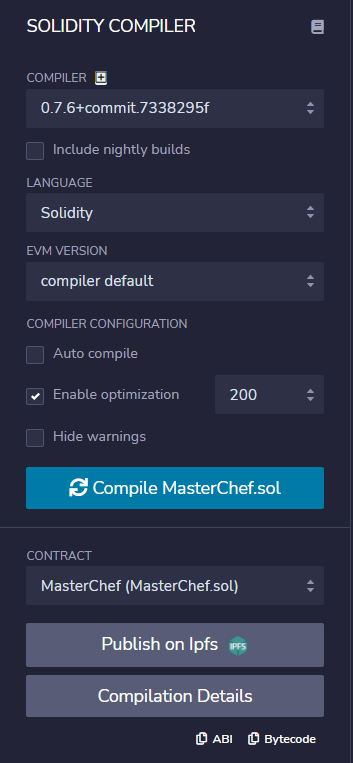
* ***Status: Open  
    
  Description:****Rewarder.sol:*There is no external manual withdraw reward function in Rewarder contract that can be called by a user. Rewards in the *Rewarder* contract are accumulated whenever the function onAPWReward is called from the *MasterChef* contract. This accumulated reward is then sent to the user as pending rewards the next time this function is called from the *MasterChef* contract by the user. There will be a situation when the user at the end would want to withdraw the pending rewards from the *Rewarder* contract without wanting to do another Deposit or Withdraw transaction in the *Masterchef* contract as that will accumulate even more rewards.  
    
  ***Recommendation:***It is recommended to add an external withdrawReward function that lets users withdraw any pending rewards in terms of Reward Tokens without having to call deposit/withdraw from the MasterChef contract. This won’t accumulate any additional rewards in the Rewarder contract.
* ***Status: Open  
    
  Description:****Rewarder.sol:  
  Rewarder* contractis missing an onlyOwner withdraw rewardToken function. This is critical as the Rewarder contract needs to always own some Reward tokens so that it can be distributed to the users. This could lead to scenarios where extra Reward tokens might get stuck in the Rewarder contract without a way to withdraw them.  
    
  ***Recommendation:***It is recommended to add an onlyOwner withdrawRewardTokens function that lets the owner withdraw any additional tokens whenever they want to.
* ***Status: Open  
    
  Description:****Rewarder.sol, line 18*:  
  The value of lpToken is never initialised but is used in lines 129 and 144.  
    
  ***Recommendation:***It is recommended to set the value of lpToken in the initialize function by adding a parameter lpToken.

# 

# Automated Audit

## Remix Compiler Warnings

All contracts were compiled successfully in remix without any errors or warnings.



## SmartCheck

Smartcheck is a tool for automated static analysis of Solidity source code for security vulnerabilities and best practices. SmartCheck translates Solidity source code into an XML-based intermediate representation and checks it against XPath patterns. Smartcheck shows significant improvements over existing alternatives in terms of false discovery rate (FDR) and false-negative rate (FNR).  
  
No major vulnerability was detected upon our analysis. Here is the report:

| ruleId: SOLIDITY\_ADDRESS\_HARDCODED  patternId: b140cd  severity: 1  line: 79  column: 44  content: 0xd9b67a26  ruleId: SOLIDITY\_ADDRESS\_HARDCODED  patternId: c67a09  severity: 1  line: 255  column: 45  content: 0x0  ruleId: SOLIDITY\_OVERPOWERED\_ROLE  patternId: j83hf7  severity: 2  line: 168  column: 4  content: functionset(uint256\_lpTokenId,uint256\_allocPoint,AmpRewarder\_rewarder,booloverwrite,bool\_withUpdate)externalonlyOwner{\_set(\_lpTokenId,\_allocPoint,\_rewarder,overwrite,\_withUpdate);}  ruleId: SOLIDITY\_OVERPOWERED\_ROLE  patternId: j83hf7  severity: 2  line: 350  column: 4  content: functionsetAPWPerBlock(uint256\_apwPerBlock)externalonlyOwner{massUpdatePools();require(\_apwPerBlock>0,"!apwPerBlock-0");apwPerBlock=\_apwPerBlock;}  ruleId: SOLIDITY\_OVERPOWERED\_ROLE  patternId: j83hf7  severity: 2  line: 362  column: 4  content: functionsetNextUpgradeAllocPoint(uint256\_lpTokenId,uint256\_nextAllocPoint)externalvalidPool(\_lpTokenId)onlyOwner{uint64ammId=lpToken.getAMMId(\_lpTokenId);uint256pairId=lpToken.getPairId(\_lpTokenId);uint256periodId=lpToken.getPeriodIndex(\_lpTokenId);require(periodId==poolToPeriodId[ammId][pairId],"Masterchef: pool already upgraded");nextUpgradeAllocPoint[\_lpTokenId]=\_nextAllocPoint;emitNextAllocPointSet(\_lpTokenId,\_nextAllocPoint);}  ruleId: SOLIDITY\_PRAGMAS\_VERSION  patternId: 23fc32  severity: 1  line: 3  column: 16  content: ^  ruleId: SOLIDITY\_PRIVATE\_MODIFIER\_DONT\_HIDE\_DATA  patternId: 5616b2  severity: 1  line: 68  column: 12  content: private  ruleId: SOLIDITY\_VISIBILITY  patternId: b51ce0  severity: 1  line: 56  column: 4  content: mapping(uint256=>PoolInfo)lpTokenIdToPoolInfo;  SOLIDITY\_VISIBILITY :1  SOLIDITY\_OVERPOWERED\_ROLE :3  SOLIDITY\_PRAGMAS\_VERSION :1  SOLIDITY\_PRIVATE\_MODIFIER\_DONT\_HIDE\_DATA :1  SOLIDITY\_ADDRESS\_HARDCODED :2 |
| --- |

## Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user’s understanding of smart contracts, assist in code reviews, and detect missing optimizations.  
  
Upon our analysis, no additional major vulnerability was detected or has been covered in the manual audit. Here is the report:

| INFO:Detectors:  OwnableUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/access/OwnableUpgradeable.sol#74) shadows:  - ContextUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/utils/ContextUpgradeable.sol#31)  ERC1155HolderUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#27) shadows:  - ERC1155ReceiverUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155ReceiverUpgradeable.sol#24)  - ERC165Upgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/introspection/ERC165Upgradeable.sol#59)  ERC1155ReceiverUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155ReceiverUpgradeable.sol#24) shadows:  - ERC165Upgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/introspection/ERC165Upgradeable.sol#59)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing  INFO:Detectors:  MasterChef.withdrawAPW(address,uint256) (amm/contracts/MasterChef.sol#357-359) ignores return value by apw.transfer(\_recipient,\_amount) (amm/contracts/MasterChef.sol#358)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer  INFO:Detectors:  AmpRewarder.lpToken (amm/contracts/Rewarder.sol#18) is never initialized. It is used in:  - AmpRewarder.pendingToken(uint256,address) (amm/contracts/Rewarder.sol#125-136)  - AmpRewarder.updatePool(uint256) (amm/contracts/Rewarder.sol#141-155)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables  INFO:Detectors:  MasterChef.pendingAPW(uint256,address) (amm/contracts/MasterChef.sol#197-212) performs a multiplication on the result of a division:  -apwReward = (block.number.sub(pool.lastRewardBlock)).mul(apwPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (amm/contracts/MasterChef.sol#205-206)  -accAPWPerShare = accAPWPerShare.add(apwReward.mul(TOKEN\_PRECISION).div(lpSupply)) (amm/contracts/MasterChef.sol#208)  MasterChef.updatePool(uint256) (amm/contracts/MasterChef.sol#223-239) performs a multiplication on the result of a division:  -pool.accAPWPerShare = pool.accAPWPerShare.add(apwReward.mul(TOKEN\_PRECISION).div(lpSupply)) (amm/contracts/MasterChef.sol#237)  -apwReward = (block.number.sub(pool.lastRewardBlock)).mul(apwPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (amm/contracts/MasterChef.sol#233-236)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply  INFO:Detectors:  Reentrancy in MasterChef.deposit(uint256,uint256) (amm/contracts/MasterChef.sol#263-304):  External calls:  - \_upgradePoolRewardsIfNeeded(\_lpTokenId) (amm/contracts/MasterChef.sol#264)  - rewarder.renewPool(previousLpTokenId,newLpTokenId) (amm/contracts/MasterChef.sol#255)  State variables written after the call(s):  - updatePool(\_lpTokenId) (amm/contracts/MasterChef.sol#272)  - pool.lastRewardBlock = block.number (amm/contracts/MasterChef.sol#230)  - pool.accAPWPerShare = pool.accAPWPerShare.add(apwReward.mul(TOKEN\_PRECISION).div(lpSupply)) (amm/contracts/MasterChef.sol#237)  - pool.lastRewardBlock = block.number (amm/contracts/MasterChef.sol#238)  Reentrancy in MasterChef.deposit(uint256,uint256) (amm/contracts/MasterChef.sol#263-304):  External calls:  - \_upgradePoolRewardsIfNeeded(\_lpTokenId) (amm/contracts/MasterChef.sol#264)  - rewarder.renewPool(previousLpTokenId,newLpTokenId) (amm/contracts/MasterChef.sol#255)  - require(bool,string)(safeAPWTransfer(msg.sender,pending),Masterchef: SafeTransfer APW failed) (amm/contracts/MasterChef.sol#285)  - success = apw.transfer(\_to,transferAmount) (amm/contracts/MasterChef.sol#346)  State variables written after the call(s):  - delete userInfo[ammId][pairId][msg.sender] (amm/contracts/MasterChef.sol#289)  - userInfo[ammId][pairId][msg.sender].periodId = periodOfToken (amm/contracts/MasterChef.sol#290)  Reentrancy in MasterChef.deposit(uint256,uint256) (amm/contracts/MasterChef.sol#263-304):  External calls:  - \_upgradePoolRewardsIfNeeded(\_lpTokenId) (amm/contracts/MasterChef.sol#264)  - rewarder.renewPool(previousLpTokenId,newLpTokenId) (amm/contracts/MasterChef.sol#255)  - require(bool,string)(safeAPWTransfer(msg.sender,pending),Masterchef: SafeTransfer APW failed) (amm/contracts/MasterChef.sol#285)  - success = apw.transfer(\_to,transferAmount) (amm/contracts/MasterChef.sol#346)  - lpToken.safeTransferFrom(address(msg.sender),address(this),\_lpTokenId,\_amount,) (amm/contracts/MasterChef.sol#293)  State variables written after the call(s):  - user.amount = user.amount.add(\_amount) (amm/contracts/MasterChef.sol#294)  - user.rewardDebt = user.amount.mul(pool.accAPWPerShare).div(TOKEN\_PRECISION) (amm/contracts/MasterChef.sol#296)Reentrancy in MasterChef.emergencyWithdraw(uint256) (amm/contracts/MasterChef.sol#328-340):  External calls:  - lpToken.safeTransferFrom(address(this),address(msg.sender),\_lpTokenId,user.amount,) (amm/contracts/MasterChef.sol#332)  State variables written after the call(s):  - user.amount = 0 (amm/contracts/MasterChef.sol#333)  - user.rewardDebt = 0 (amm/contracts/MasterChef.sol#334)  Reentrancy in AmpRewarder.onAPWReward(uint256,address,address,uint256) (amm/contracts/Rewarder.sol#63-79):  External calls:  - rewardToken.safeTransfer(to,pending) (amm/contracts/Rewarder.sol#74)  State variables written after the call(s):  - user.amount = apwAmount (amm/contracts/Rewarder.sol#76)  - user.rewardDebt = apwAmount.mul(pool.accAPWPerShare) / ACC\_TOKEN\_PRECISION (amm/contracts/Rewarder.sol#77)  Reentrancy in MasterChef.withdraw(uint256,uint256) (amm/contracts/MasterChef.sol#307-325):  External calls:  - require(bool,string)(safeAPWTransfer(msg.sender,pending),Masterchef: SafeTransfer APW failed) (amm/contracts/MasterChef.sol#315)  - success = apw.transfer(\_to,transferAmount) (amm/contracts/MasterChef.sol#346)  State variables written after the call(s):  - user.amount = user.amount.sub(\_amount) (amm/contracts/MasterChef.sol#316)  - user.rewardDebt = user.amount.mul(pool.accAPWPerShare).div(TOKEN\_PRECISION) (amm/contracts/MasterChef.sol#318)Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1  INFO:Detectors:  AmpRewarder.onAPWReward(uint256,address,address,uint256).pending (amm/contracts/Rewarder.sol#71) is a local variable never initialized  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables  INFO:Detectors:  MasterChef.\_add(uint256,uint256,AmpRewarder,bool) (amm/contracts/MasterChef.sol#130-161) ignores return value by activePools.add(\_lpTokenId) (amm/contracts/MasterChef.sol#159)  MasterChef.\_upgradePoolRewardsIfNeeded(uint256) (amm/contracts/MasterChef.sol#242-260) ignores return value by activePools.remove(previousLpTokenId) (amm/contracts/MasterChef.sol#252)  MasterChef.deposit(uint256,uint256) (amm/contracts/MasterChef.sol#263-304) ignores return value by userLpTokensIds[msg.sender].remove(IAMM(lpToken.amms(ammId)).getLPTokenId(ammId,pairId,user.periodId)) (amm/contracts/MasterChef.sol#288)  MasterChef.deposit(uint256,uint256) (amm/contracts/MasterChef.sol#263-304) ignores return value by userLpTokensIds[msg.sender].add(\_lpTokenId) (amm/contracts/MasterChef.sol#295)  MasterChef.withdraw(uint256,uint256) (amm/contracts/MasterChef.sol#307-325) ignores return value by userLpTokensIds[msg.sender].remove(\_lpTokenId) (amm/contracts/MasterChef.sol#317)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return  INFO:Detectors:  Reentrancy in MasterChef.deposit(uint256,uint256) (amm/contracts/MasterChef.sol#263-304):  External calls:  - \_upgradePoolRewardsIfNeeded(\_lpTokenId) (amm/contracts/MasterChef.sol#264)  - rewarder.renewPool(previousLpTokenId,newLpTokenId) (amm/contracts/MasterChef.sol#255)  - require(bool,string)(safeAPWTransfer(msg.sender,pending),Masterchef: SafeTransfer APW failed) (amm/contracts/MasterChef.sol#285)  - success = apw.transfer(\_to,transferAmount) (amm/contracts/MasterChef.sol#346)  - lpToken.safeTransferFrom(address(msg.sender),address(this),\_lpTokenId,\_amount,) (amm/contracts/MasterChef.sol#293)  - \_rewarder.onAPWReward(\_lpTokenId,msg.sender,msg.sender,user.amount) (amm/contracts/MasterChef.sol#300)  Event emitted after the call(s):  - Deposit(msg.sender,\_lpTokenId,\_amount) (amm/contracts/MasterChef.sol#303)  Reentrancy in MasterChef.emergencyWithdraw(uint256) (amm/contracts/MasterChef.sol#328-340):  External calls:  - lpToken.safeTransferFrom(address(this),address(msg.sender),\_lpTokenId,user.amount,) (amm/contracts/MasterChef.sol#332)  - \_rewarder.onAPWReward(\_lpTokenId,msg.sender,msg.sender,0) (amm/contracts/MasterChef.sol#337)  Event emitted after the call(s):  - EmergencyWithdraw(msg.sender,\_lpTokenId,user.amount) (amm/contracts/MasterChef.sol#339)  Reentrancy in AmpRewarder.onAPWReward(uint256,address,address,uint256) (amm/contracts/Rewarder.sol#63-79):  External calls:  - rewardToken.safeTransfer(to,pending) (amm/contracts/Rewarder.sol#74)  Event emitted after the call(s):  - LogOnReward(\_user,pid,pending,to) (amm/contracts/Rewarder.sol#78)  Reentrancy in MasterChef.withdraw(uint256,uint256) (amm/contracts/MasterChef.sol#307-325):  External calls:  - require(bool,string)(safeAPWTransfer(msg.sender,pending),Masterchef: SafeTransfer APW failed) (amm/contracts/MasterChef.sol#315)  - success = apw.transfer(\_to,transferAmount) (amm/contracts/MasterChef.sol#346)  - \_rewarder.onAPWReward(\_lpTokenId,msg.sender,msg.sender,user.amount) (amm/contracts/MasterChef.sol#321)  - lpToken.safeTransferFrom(address(this),address(msg.sender),\_lpTokenId,\_amount,) (amm/contracts/MasterChef.sol#323)  Event emitted after the call(s):  - Withdraw(msg.sender,\_lpTokenId,\_amount) (amm/contracts/MasterChef.sol#324)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3  INFO:Detectors:  AmpRewarder.pendingToken(uint256,address) (amm/contracts/Rewarder.sol#125-136) uses timestamp for comparisons  Dangerous comparisons:  - block.timestamp > pool.lastRewardTime && lpSupply != 0 (amm/contracts/Rewarder.sol#130)  AmpRewarder.updatePool(uint256) (amm/contracts/Rewarder.sol#141-155) uses timestamp for comparisons  Dangerous comparisons:  - block.timestamp > pool.lastRewardTime (amm/contracts/Rewarder.sol#143)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp  INFO:Detectors:  AddressUpgradeable.isContract(address) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#26-35) uses assembly  - INLINE ASM (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#33)  AddressUpgradeable.\_verifyCallResult(bool,bytes,string) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#147-164) uses assembly  - INLINE ASM (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#156-159)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage  INFO:Detectors:  Different versions of Solidity is used:  - Version used: ['0.7.6', '>=0.4.24<0.8.0', '>=0.6.0<0.8.0', '^0.7.0', '^0.7.6']  - ^0.7.6 (amm/contracts/MasterChef.sol#3)  - v2 (amm/contracts/Rewarder.sol#2)  - ^0.7.6 (amm/contracts/Rewarder.sol#3)  - 0.7.6 (amm/contracts/interfaces/IAMM.sol#3)  - v2 (amm/contracts/interfaces/IAMM.sol#4)  - 0.7.6 (amm/contracts/interfaces/IERC1155.sol#3)  - 0.7.6 (amm/contracts/interfaces/IERC20.sol#3)  - 0.7.6 (amm/contracts/interfaces/IFutureVault.sol#3)  - 0.7.6 (amm/contracts/interfaces/IFutureWallet.sol#3)  - ^0.7.6 (amm/contracts/interfaces/ILPToken.sol#5)  - 0.7.6 (amm/contracts/interfaces/IPT.sol#3)  - 0.7.6 (amm/contracts/interfaces/IRegistry.sol#3)  - ABIEncoderV2 (amm/contracts/interfaces/IRegistry.sol#4)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/access/OwnableUpgradeable.sol#3)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/introspection/ERC165Upgradeable.sol#3)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/introspection/IERC165Upgradeable.sol#3)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#3)  - >=0.4.24<0.8.0 (amm/contracts/openzeppelin/contracts/proxy/Initializable.sol#4)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#3)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155ReceiverUpgradeable.sol#3)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/token/ERC1155/IERC1155ReceiverUpgradeable.sol#3)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/token/ERC20/IERC20Upgradeable.sol#3)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/token/ERC20/SafeERC20Upgradeable.sol#3)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#3)  - >=0.6.0<0.8.0 (amm/contracts/openzeppelin/contracts/utils/ContextUpgradeable.sol#3)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#3)  - ^0.7.0 (amm/contracts/openzeppelin/contracts/utils/ReentrancyGuardUpgradeable.sol#3)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used  INFO:Detectors:  AddressUpgradeable.functionCall(address,bytes) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#79-81) is never used and should be removed  AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#104-106) is never used and should be removed  AddressUpgradeable.functionStaticCall(address,bytes) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#129-131) is never used and should be removed  AddressUpgradeable.functionStaticCall(address,bytes,string) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#139-145) is never used and should be removed  AddressUpgradeable.sendValue(address,uint256) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#53-59) is never used and should be removed  ContextUpgradeable.\_\_Context\_init() (amm/contracts/openzeppelin/contracts/utils/ContextUpgradeable.sol#17-19) is never used and should be removed  ContextUpgradeable.\_msgData() (amm/contracts/openzeppelin/contracts/utils/ContextUpgradeable.sol#27-30) is never used and should be removed  ERC1155HolderUpgradeable.\_\_ERC1155Holder\_init() (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#12-16) is never used and should be removed  ERC1155HolderUpgradeable.\_\_ERC1155Holder\_init\_unchained() (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#18-19) is never used and should be removed  ERC1155ReceiverUpgradeable.\_\_ERC1155Receiver\_init() (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155ReceiverUpgradeable.sol#13-16) is never used and should be removed  ERC1155ReceiverUpgradeable.\_\_ERC1155Receiver\_init\_unchained() (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155ReceiverUpgradeable.sol#18-23) is never used and should be removed  ERC165Upgradeable.\_\_ERC165\_init() (amm/contracts/openzeppelin/contracts/introspection/ERC165Upgradeable.sol#25-27) is never used and should be removed  ERC165Upgradeable.\_\_ERC165\_init\_unchained() (amm/contracts/openzeppelin/contracts/introspection/ERC165Upgradeable.sol#29-33) is never used and should be removed  EnumerableSetUpgradeable.add(EnumerableSetUpgradeable.AddressSet,address) (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#201-203) is never used and should be removed  EnumerableSetUpgradeable.add(EnumerableSetUpgradeable.Bytes32Set,bytes32) (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#147-149) is never used and should be removed  EnumerableSetUpgradeable.at(EnumerableSetUpgradeable.AddressSet,uint256) (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#239-241) is never used and should be removed  EnumerableSetUpgradeable.at(EnumerableSetUpgradeable.Bytes32Set,uint256) (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#185-187) is never used and should be removed  EnumerableSetUpgradeable.contains(EnumerableSetUpgradeable.AddressSet,address) (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#218-220) is never used and should be removed  EnumerableSetUpgradeable.contains(EnumerableSetUpgradeable.Bytes32Set,bytes32) (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#164-166) is never used and should be removed  EnumerableSetUpgradeable.length(EnumerableSetUpgradeable.AddressSet) (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#225-227) is never used and should be removed  EnumerableSetUpgradeable.length(EnumerableSetUpgradeable.Bytes32Set) (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#171-173) is never used and should be removed  EnumerableSetUpgradeable.remove(EnumerableSetUpgradeable.AddressSet,address) (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#211-213) is never used and should be removed  EnumerableSetUpgradeable.remove(EnumerableSetUpgradeable.Bytes32Set,bytes32) (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#157-159) is never used and should be removed  ReentrancyGuardUpgradeable.\_\_ReentrancyGuard\_init() (amm/contracts/openzeppelin/contracts/utils/ReentrancyGuardUpgradeable.sol#39-41) is never used and should be removed  ReentrancyGuardUpgradeable.\_\_ReentrancyGuard\_init\_unchained() (amm/contracts/openzeppelin/contracts/utils/ReentrancyGuardUpgradeable.sol#43-45) is never used and should be removed  SafeERC20Upgradeable.safeApprove(IERC20Upgradeable,address,uint256) (amm/contracts/openzeppelin/contracts/token/ERC20/SafeERC20Upgradeable.sol#37-46) is never used and should be removed  SafeERC20Upgradeable.safeDecreaseAllowance(IERC20Upgradeable,address,uint256) (amm/contracts/openzeppelin/contracts/token/ERC20/SafeERC20Upgradeable.sol#53-56) is never used and should be removed  SafeERC20Upgradeable.safeIncreaseAllowance(IERC20Upgradeable,address,uint256) (amm/contracts/openzeppelin/contracts/token/ERC20/SafeERC20Upgradeable.sol#48-51) is never used and should be removed  SafeERC20Upgradeable.safeTransferFrom(IERC20Upgradeable,address,address,uint256) (amm/contracts/openzeppelin/contracts/token/ERC20/SafeERC20Upgradeable.sol#26-28) is never used and should be removed  SafeMathUpgradeable.div(uint256,uint256,string) (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#190-193) is never used and should be removed  SafeMathUpgradeable.mod(uint256,uint256) (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#152-155) is never used and should be removed  SafeMathUpgradeable.mod(uint256,uint256,string) (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#210-213) is never used and should be removed  SafeMathUpgradeable.sub(uint256,uint256,string) (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#170-173) is never used and should be removed  SafeMathUpgradeable.tryAdd(uint256,uint256) (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#24-28) is never used and should be removed  SafeMathUpgradeable.tryDiv(uint256,uint256) (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#60-63) is never used and should be removed  SafeMathUpgradeable.tryMod(uint256,uint256) (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#70-73) is never used and should be removed  SafeMathUpgradeable.tryMul(uint256,uint256) (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#45-53) is never used and should be removed  SafeMathUpgradeable.trySub(uint256,uint256) (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#35-38) is never used and should be removed  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code  INFO:Detectors:  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/access/OwnableUpgradeable.sol#3) allows old versions  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/introspection/ERC165Upgradeable.sol#3) allows old versions  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/introspection/IERC165Upgradeable.sol#3) allows old versions  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/math/SafeMathUpgradeable.sol#3) allows old versions  Pragma version>=0.4.24<0.8.0 (amm/contracts/openzeppelin/contracts/proxy/Initializable.sol#4) is too complex  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#3) allows old versions  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155ReceiverUpgradeable.sol#3) allows old versions  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/token/ERC1155/IERC1155ReceiverUpgradeable.sol#3) allows old versions  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/token/ERC20/IERC20Upgradeable.sol#3) allows old versions  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/token/ERC20/SafeERC20Upgradeable.sol#3) allows old versions  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#3) allows old versions  Pragma version>=0.6.0<0.8.0 (amm/contracts/openzeppelin/contracts/utils/ContextUpgradeable.sol#3) is too complex  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/utils/EnumerableSetUpgradeable.sol#3) allows old versions  Pragma version^0.7.0 (amm/contracts/openzeppelin/contracts/utils/ReentrancyGuardUpgradeable.sol#3) allows old versions  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity  INFO:Detectors:  Low level call in AddressUpgradeable.sendValue(address,uint256) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#53-59):  - (success) = recipient.call{value: amount}() (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#57)  Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#114-121):  - (success,returndata) = target.call{value: value}(data) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#119)  Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#139-145):  - (success,returndata) = target.staticcall(data) (amm/contracts/openzeppelin/contracts/utils/AddressUpgradeable.sol#143)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls  INFO:Detectors:  Parameter MasterChef.initialize(address,address,uint256,uint256).\_apw (amm/contracts/MasterChef.sol#101) is not in mixedCase  Parameter MasterChef.initialize(address,address,uint256,uint256).\_lpToken (amm/contracts/MasterChef.sol#102) is not in mixedCase  Parameter MasterChef.initialize(address,address,uint256,uint256).\_apwPerBlock (amm/contracts/MasterChef.sol#103) is not in mixedCase  Parameter MasterChef.initialize(address,address,uint256,uint256).\_startBlock (amm/contracts/MasterChef.sol#104) is not in mixedCase  Parameter MasterChef.add(uint256,uint256,AmpRewarder,bool).\_allocPoint (amm/contracts/MasterChef.sol#121) is not in mixedCase  Parameter MasterChef.add(uint256,uint256,AmpRewarder,bool).\_lpTokenId (amm/contracts/MasterChef.sol#122) is not in mixedCase  Parameter MasterChef.add(uint256,uint256,AmpRewarder,bool).\_rewarder (amm/contracts/MasterChef.sol#123) is not in mixedCase  Parameter MasterChef.add(uint256,uint256,AmpRewarder,bool).\_withUpdate (amm/contracts/MasterChef.sol#124) is not in mixedCase  Parameter MasterChef.set(uint256,uint256,AmpRewarder,bool,bool).\_lpTokenId (amm/contracts/MasterChef.sol#169) is not in mixedCase  Parameter MasterChef.set(uint256,uint256,AmpRewarder,bool,bool).\_allocPoint (amm/contracts/MasterChef.sol#170) is not in mixedCase  Parameter MasterChef.set(uint256,uint256,AmpRewarder,bool,bool).\_rewarder (amm/contracts/MasterChef.sol#171) is not in mixedCase  Parameter MasterChef.set(uint256,uint256,AmpRewarder,bool,bool).\_withUpdate (amm/contracts/MasterChef.sol#173) is not in mixedCase  Parameter MasterChef.pendingAPW(uint256,address).\_lpTokenId (amm/contracts/MasterChef.sol#197) is not in mixedCase  Parameter MasterChef.pendingAPW(uint256,address).\_user (amm/contracts/MasterChef.sol#197) is not in mixedCase  Parameter MasterChef.updatePool(uint256).\_lpTokenId (amm/contracts/MasterChef.sol#223) is not in mixedCase  Parameter MasterChef.deposit(uint256,uint256).\_lpTokenId (amm/contracts/MasterChef.sol#263) is not in mixedCase  Parameter MasterChef.deposit(uint256,uint256).\_amount (amm/contracts/MasterChef.sol#263) is not in mixedCase  Parameter MasterChef.withdraw(uint256,uint256).\_lpTokenId (amm/contracts/MasterChef.sol#307) is not in mixedCase  Parameter MasterChef.withdraw(uint256,uint256).\_amount (amm/contracts/MasterChef.sol#307) is not in mixedCase  Parameter MasterChef.emergencyWithdraw(uint256).\_lpTokenId (amm/contracts/MasterChef.sol#328) is not in mixedCase  Parameter MasterChef.safeAPWTransfer(address,uint256).\_to (amm/contracts/MasterChef.sol#343) is not in mixedCase  Parameter MasterChef.safeAPWTransfer(address,uint256).\_amount (amm/contracts/MasterChef.sol#343) is not in mixedCase  Parameter MasterChef.setAPWPerBlock(uint256).\_apwPerBlock (amm/contracts/MasterChef.sol#350) is not in mixedCase  Parameter MasterChef.withdrawAPW(address,uint256).\_recipient (amm/contracts/MasterChef.sol#357) is not in mixedCase  Parameter MasterChef.withdrawAPW(address,uint256).\_amount (amm/contracts/MasterChef.sol#357) is not in mixedCase  Parameter MasterChef.setNextUpgradeAllocPoint(uint256,uint256).\_lpTokenId (amm/contracts/MasterChef.sol#362) is not in mixedCase  Parameter MasterChef.setNextUpgradeAllocPoint(uint256,uint256).\_nextAllocPoint (amm/contracts/MasterChef.sol#362) is not in mixedCase  Parameter MasterChef.isRegisteredPoolId(uint256).\_poolId (amm/contracts/MasterChef.sol#371) is not in mixedCase  Parameter MasterChef.poolIdAt(uint256).\_id (amm/contracts/MasterChef.sol#379) is not in mixedCase  Parameter MasterChef.getUserLpTokenIdList(address).\_user (amm/contracts/MasterChef.sol#383) is not in mixedCase  Parameter AmpRewarder.initialize(address,IERC20).\_MASTERCHEF\_V2 (amm/contracts/Rewarder.sol#55) is not in mixedCase  Parameter AmpRewarder.initialize(address,IERC20).\_rewardToken (amm/contracts/Rewarder.sol#55) is not in mixedCase  Parameter AmpRewarder.onAPWReward(uint256,address,address,uint256).\_user (amm/contracts/Rewarder.sol#65) is not in mixedCase  Parameter AmpRewarder.setRewardPerSecond(uint256,uint256).\_pid (amm/contracts/Rewarder.sol#96) is not in mixedCase  Parameter AmpRewarder.setRewardPerSecond(uint256,uint256).\_rewardPerSecond (amm/contracts/Rewarder.sol#96) is not in mixedCase  Parameter AmpRewarder.renewPool(uint256,uint256).\_oldPid (amm/contracts/Rewarder.sol#100) is not in mixedCase  Parameter AmpRewarder.renewPool(uint256,uint256).\_newPid (amm/contracts/Rewarder.sol#100) is not in mixedCase  Parameter AmpRewarder.setNextRewardPerSecond(uint256,uint256).\_pid (amm/contracts/Rewarder.sol#106) is not in mixedCase  Parameter AmpRewarder.setNextRewardPerSecond(uint256,uint256).\_nextRewardPerSecond (amm/contracts/Rewarder.sol#106) is not in mixedCase  Parameter AmpRewarder.pendingToken(uint256,address).\_pid (amm/contracts/Rewarder.sol#125) is not in mixedCase  Parameter AmpRewarder.pendingToken(uint256,address).\_user (amm/contracts/Rewarder.sol#125) is not in mixedCase  Variable AmpRewarder.MASTERCHEF\_V2 (amm/contracts/Rewarder.sol#46) is not in mixedCase  Function IERC1155.MINTER\_ROLE() (amm/contracts/interfaces/IERC1155.sol#118) is not in mixedCase  Function IFutureVault.PERIOD\_DURATION() (amm/contracts/interfaces/IFutureVault.sol#27) is not in mixedCase  Function IFutureVault.PLATFORM\_NAME() (amm/contracts/interfaces/IFutureVault.sol#33) is not in mixedCase  Parameter IRegistry.setPTLogic(address).\_PTLogic (amm/contracts/interfaces/IRegistry.sol#42) is not in mixedCase  Parameter IRegistry.setFYTLogic(address).\_FYTLogic (amm/contracts/interfaces/IRegistry.sol#48) is not in mixedCase  Function OwnableUpgradeable.\_\_Ownable\_init() (amm/contracts/openzeppelin/contracts/access/OwnableUpgradeable.sol#27-30) is not in mixedCase  Function OwnableUpgradeable.\_\_Ownable\_init\_unchained() (amm/contracts/openzeppelin/contracts/access/OwnableUpgradeable.sol#32-36) is not in mixedCase  Variable OwnableUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/access/OwnableUpgradeable.sol#74) is not in mixedCase  Function ERC165Upgradeable.\_\_ERC165\_init() (amm/contracts/openzeppelin/contracts/introspection/ERC165Upgradeable.sol#25-27) is not in mixedCase  Function ERC165Upgradeable.\_\_ERC165\_init\_unchained() (amm/contracts/openzeppelin/contracts/introspection/ERC165Upgradeable.sol#29-33) is not in mixedCase  Variable ERC165Upgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/introspection/ERC165Upgradeable.sol#59) is not in mixedCase  Function ERC1155HolderUpgradeable.\_\_ERC1155Holder\_init() (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#12-16) is not in mixedCase  Function ERC1155HolderUpgradeable.\_\_ERC1155Holder\_init\_unchained() (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#18-19) is not in mixedCase  Variable ERC1155HolderUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#27) is not in mixedCase  Function ERC1155ReceiverUpgradeable.\_\_ERC1155Receiver\_init() (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155ReceiverUpgradeable.sol#13-16) is not in mixedCase  Function ERC1155ReceiverUpgradeable.\_\_ERC1155Receiver\_init\_unchained() (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155ReceiverUpgradeable.sol#18-23) is not in mixedCase  Variable ERC1155ReceiverUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155ReceiverUpgradeable.sol#24) is not in mixedCase  Function ContextUpgradeable.\_\_Context\_init() (amm/contracts/openzeppelin/contracts/utils/ContextUpgradeable.sol#17-19) is not in mixedCase  Function ContextUpgradeable.\_\_Context\_init\_unchained() (amm/contracts/openzeppelin/contracts/utils/ContextUpgradeable.sol#21-22) is not in mixedCase  Variable ContextUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/utils/ContextUpgradeable.sol#31) is not in mixedCase  Function ReentrancyGuardUpgradeable.\_\_ReentrancyGuard\_init() (amm/contracts/openzeppelin/contracts/utils/ReentrancyGuardUpgradeable.sol#39-41) is not in mixedCase  Function ReentrancyGuardUpgradeable.\_\_ReentrancyGuard\_init\_unchained() (amm/contracts/openzeppelin/contracts/utils/ReentrancyGuardUpgradeable.sol#43-45) is not in mixedCase  Variable ReentrancyGuardUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/utils/ReentrancyGuardUpgradeable.sol#67) is not in mixedCase  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions  INFO:Detectors:  Redundant expression "this (amm/contracts/openzeppelin/contracts/utils/ContextUpgradeable.sol#28)" inContextUpgradeable (amm/contracts/openzeppelin/contracts/utils/ContextUpgradeable.sol#16-32)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements  INFO:Detectors:  Variable MasterChef.getUserLpTokenIdList(address).\_userLpTokenIds (amm/contracts/MasterChef.sol#385) is too similar to MasterChef.userLpTokensIds (amm/contracts/MasterChef.sol#81)  Variable AmpRewarder.initialize(address,IERC20).\_rewardToken (amm/contracts/Rewarder.sol#55) is too similar to AmpRewarder.pendingTokens(uint256,address).rewardTokens (amm/contracts/Rewarder.sol#84)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar  INFO:Detectors:  ERC1155HolderUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#27) is never used in MasterChef (amm/contracts/MasterChef.sol#24-391)  MasterChef.lpTokenIdToPoolInfo (amm/contracts/MasterChef.sol#56) is never used in MasterChef (amm/contracts/MasterChef.sol#24-391)  OwnableUpgradeable.\_\_gap (amm/contracts/openzeppelin/contracts/access/OwnableUpgradeable.sol#74) is never used in AmpRewarder (amm/contracts/Rewarder.sol#11-156)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables  INFO:Detectors:  initialize(address,address,uint256,uint256) should be declared external:  - MasterChef.initialize(address,address,uint256,uint256) (amm/contracts/MasterChef.sol#100-116)  initialize(address,IERC20) should be declared external:  - AmpRewarder.initialize(address,IERC20) (amm/contracts/Rewarder.sol#55-61)  setRewardPerSecond(uint256,uint256) should be declared external:  - AmpRewarder.setRewardPerSecond(uint256,uint256) (amm/contracts/Rewarder.sol#96-98)  renewPool(uint256,uint256) should be declared external:  - AmpRewarder.renewPool(uint256,uint256) (amm/contracts/Rewarder.sol#100-104)  setNextRewardPerSecond(uint256,uint256) should be declared external:  - AmpRewarder.setNextRewardPerSecond(uint256,uint256) (amm/contracts/Rewarder.sol#106-109)  renounceOwnership() should be declared external:  - OwnableUpgradeable.renounceOwnership() (amm/contracts/openzeppelin/contracts/access/OwnableUpgradeable.sol#60-63)  transferOwnership(address) should be declared external:  - OwnableUpgradeable.transferOwnership(address) (amm/contracts/openzeppelin/contracts/access/OwnableUpgradeable.sol#69-73)  supportsInterface(bytes4) should be declared external:  - ERC165Upgradeable.supportsInterface(bytes4) (amm/contracts/openzeppelin/contracts/introspection/ERC165Upgradeable.sol#40-42)  onERC1155Received(address,address,uint256,uint256,bytes) should be declared external:  - ERC1155HolderUpgradeable.onERC1155Received(address,address,uint256,uint256,bytes) (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#20-22)  onERC1155BatchReceived(address,address,uint256[],uint256[],bytes) should be declared external:  - ERC1155HolderUpgradeable.onERC1155BatchReceived(address,address,uint256[],uint256[],bytes) (amm/contracts/openzeppelin/contracts/token/ERC1155/ERC1155HolderUpgradeable.sol#24-26)  Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-externalINFO:Slither:amm/contracts/MasterChef.sol analyzed (24 contracts with 75 detectors), 164 result(s) found |
| --- |

# Disclaimer

This audit is not a security warranty, investment advice, or an endorsement of the APWine MasterChef smart contracts. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

# Summary

Altogether, the code is well written and demonstrates effective use of abstraction, separation of concerns, and modularity. There are a few high severity issues found which need to be closed upon code modification or use case discussion. The MasterChef and Rewarder contracts are recommended to be deployed on mainnet only after the high severity issues are resolved. It is also recommended that the medium and low severity issues are also closed upon code modifications or discussions