PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For YieldWolf

23 September 2021

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

1 Overview

This report has been prepared for YieldWolf on the Polygon network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	YieldWolf
URL	https://yieldwolf.finance
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
YieldWolf	0xbf65023bcf48ad0ab5537ea39c9242de499386c9	MATCH
AutoCompoundStrategy	Inherited by ACMasterChef	MATCH
ACMiniChef	0x250B171274beFa912C0993FB037705920bB5147b	MATCH
ACMasterChefWithStaking	0x7De84ee43c4f7bA0487B77cD5Dd6f2D15903b33d	MATCH
ACMasterChefWithRef	ACMasterChefWithRef.sol	PENDING
ACMasterChef	0x6D879b9c3B85F2Fe42e29318bdFA15055c6FA787	MATCH
NativeValueLowerThanCondition	Will not use in production	PENDING
SimpleWithdrawAction	SimpleWithdrawAction.sol	PENDING

The client has said that would deploy ACMasterChefWithRef when they need to integrate a farm that requires this interface, and that SimpleWithdrawAction is meant to be deployed by users to use rules.

For reference, the files that we have audited can be found in this GitHub commit: https://github.com/yieldwolf/contracts/tree/6cc8b5d2d58ed81832b2ee7a3da097da4b59e7c2/contracts/strategies

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	7	7	-	-
Medium	4	4	-	-
Low	8	7	-	1
Informational	9	7	-	2
Total	**28**	**25**	**-**	**3**

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 YieldWolf

ID	Severity	Summary	Status
01	MEDIUM	`ruleFeeAmount` is based on the percentage of the user's staked amount, instead of the withdraw amount	RESOLVED
02	MEDIUM	Lack of ability to remove added rules for an account	RESOLVED
03	LOW	`depositTo` allows to deposit for another account, allow phishing attacks on users	ACKNOWLEDGED
04	LOW	`setRuleFee` is checking against `PERFORMANCE_FEE_CAP` instead of `RULE_EXECUTION_FEE_CAP`	RESOLVED
05	LOW	Lack of rule and action contract validation	RESOLVED
06	LOW	Lack of sanity check for `_stakeToken` and `_strategy` when adding a new pool	RESOLVED
07	LOW	Reentrancy guard is not done on `withdrawFrom`	RESOLVED
08	INFO	Parameter array length checks are not done in `addMany`	RESOLVED
09	INFO	Lack of cap for number of rules for each account in a pool	RESOLVED
10	INFO	`executeRule` should be limited to EOA only to reduce the attack surface	RESOLVED

1.4.2 AutoCompoundStrategy

ID	Severity	Summary	Status
11	HIGH	`earnAmount` might not equate to the same token amount harvested, but balance of the strategy contract	RESOLVED
12	HIGH	Multiple `emergencyWithdraw` calls on underlying farms with deposit / withdraw fees, or uses a fee on transfer token	RESOLVED
13	HIGH	`totalValueLockedNative` uses LP underlying balances to calculate the pricing	RESOLVED
14	HIGH	`setExtraEarnTokens` has no maximum length for `extraEarnTokens`, which can result in denial of service for the `earn` function	RESOLVED
15	HIGH	SwapRouter can be modified by the owner	RESOLVED
16	MEDIUM	Lack of validation of `extraEarnTokens` in `setExtraEarnTokens`	RESOLVED
17	MEDIUM	`farm()` does not check if contract is paused	RESOLVED
18	LOW	`tokenToEarn` lacks sufficient token validation	RESOLVED
19	LOW	`tokenToEarn` reverts if earnToken is a token attempted to be swapped	RESOLVED
20	LOW	`tokenToEarn` can fail if the token does not have a native token pair, or the liquidity pool has insufficient liquidity for a successful swap	RESOLVED
21	INFO	`block.timestamp` can be used as the deadline for swaps and adding liquidity	RESOLVED
22	INFO	Swap and liquidity router can be different addresses	ACKNOWLEDGED
23	INFO	Token 0 and 1 should be obtained from the LP contract if stake token is an LP	RESOLVED
24	INFO	`setReferrer` allows setting strategy's referrer in the underlying farm	ACKNOWLEDGED
25	INFO	`earn` may fail if the underlying farm prevents harvesting with 0 as the withdraw amount	RESOLVED

1.4.3 NativeValueLowerThanCondition

ID	Severity	Summary	Status
26	HIGH	Rule is vulnerable to price manipulation attacks	RESOLVED
27	HIGH	Improper calculation of `userNativeValue`	RESOLVED

1.4.4 SimpleWithdrawAction

ID	Severity	Summary	Status
28	INFO	Action's and callback should not need to pass the `_yieldWolf` parameter	RESOLVED

2 Findings

2.1 YieldWolf

Overview

The YieldWolf contract is the main vault in which users deposit their tokens, which will then be deposited into the respective strategies. The deposited tokens will then compound over time using the yield gained from the underlying farm.

On each deposit or withdrawal, the `earn` function is attempted which will make the strategy harvest the yield and convert it to more of the deposit token. `emergencyWithdraw` will skip calling the `earn` function.

Fees

Deposits and withdrawals can be subject to fees, and there is also a performance fee, rule fee, and bounty fees for the performance and rule fees. All these fees can be modified by the owner of the contract. Deposit and withdrawal fees are capped at 5%, and performance fees and rule fees are capped at 5%. Bounty fees can be set up to a maximum of 100% of the performance and rule fees.

Fee on transfer tokens are supported, however, it is to be noted that there will be two transfers done on deposits and withdrawals. Here is an example of a deposit done for 100 tokens with a 5% fee on transfer into a strategy that has a 5% deposit fee:

The vault will call a `safeTransferFrom` of 100 tokens from the user to the underlying strategy, which will receive 95 tokens. The vault will then call `strategy.deposit(95 tokens)`. In the strategy, deposit fees will be charged as 5% of 95 tokens, which will be 95 - 4.75 = 90.25 tokens. The strategy then deposits the balance to the farm, and then calculates the actual deposit as such:

`totalStakedBefore` = (x + 90.25) - 90.25 tokens

Deposits 90.25 tokens into the underlying farm

`totalStakedAfter` = x + (90.25 *0.95) tokens

`_depositAmount` = x + 85.37 - (x) = 85.37 tokens

This will result in 85.37 tokens from an initial 100 token deposit.

For performance fees, the `earn` function can be called by anyone and the caller can earn a bounty which is a percentage of the performance fees.

Rules and actions

One additional feature added in this vault contract is the concept of rules. Users can add rules to a position (deposit in a pool id) which can then be executed by anyone who can earn a bounty from successful execution. Once a rule's criteria is met, it can then do a withdrawal of a portion of the user's deposit in the strategy, which can then be deposited to an Action contract (specified by `withdrawTo` which is defined by the Rule contract). The callback function in the Action contract will then be executed and some actions such as swapping tokens can be done, as defined in the callback function.

By default, accounts that deposit into the vault do not have any rules, and only add rules for their own account. It should be noted that these rules should only be added by users who know what they are doing, as improper implementation of it can result in a loss of user funds.

2.1.1 Privileged Roles and Actions

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

- `setOperator`
- `setDepositFee`
- `setWithdrawFee`
- `setPerformanceFee`
- `setPerformanceFeeBountyPct`
- `setRuleFee`
- `setRuleFeeBountyPct`
- `setStrategySwapRouter`
- `setStrategySwapPath`
- `setStrategyExtraEarnTokens`

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

- `add`
- `addMany`
- `feeAddressSetter`
- `setFeeAddress`
- `setFeeAddressSetter`

2.1.2 Issues & Recommendations

Issue #01	`ruleFeeAmount` is based on the percentage of the user's staked amount, instead of the withdraw amount
Severity	MEDIUM SEVERITY

Description	In executeRule, the ruleFeeAmount is calculated based on the user's staked token balance, instead of the withdrawAmount.
	
	`(uint256 withdrawAmount, address withdrawTo) = action.execute(`
	`  address(this),`
	`  address(strategy),`
	`  _user,`
	`  _pid,`
	`  rule.actionIntInputs,`
	`  rule.actionAddrInputs`
	`);`
	
	`uint256 staked = stakedTokens(_pid, _user);`
	`uint256 ruleFeeAmount = (staked * ruleFee) / 10000;`
	`if (withdrawAmount < ruleFeeAmount) {`
	`  withdrawAmount = ruleFeeAmount;`
	`}`
	
	If a rule is executed, even if the actual `withdrawAmount` is lesser in comparison to the user's staked balance, the amount tax is based on the much larger amount, and could result in unexpected losses for the user using the rule. One thing to note is that the `ruleFeeAmount` can never be greater than the `withdrawAmount`.

Recommendation	Consider changing the calculation of the `ruleFee` based on the `withdrawAmount` or `stakedToken` amount, whichever is lesser.
Resolution	RESOLVED The `ruleFee` has been changed to be based on `withdrawAmount` or `stakedToken` amount, whichever is lesser.

Issue #02	Lack of ability to remove added rules for an account
Severity	MEDIUM SEVERITY
Description	An account would be able to add new rules, but there is no way to remove a rule. This could result in some security related issues and loss of funds if an added rule has a flaw or is mis-implemented.
Recommendation	Add a function to allow an account to remove added rules.
Resolution	RESOLVED A `removeRule` function has been added.

Issue #03	`depositTo` allows to deposit for another account, allow phishing attacks on users
Severity	LOW SEVERITY
Description	The user can be tricked into signing a tx with `depositTo`'s `_to` as an attacker's address. This could result in a loss of funds as funds would be depositfor the attacker's address, which the attacker can then withdraw. Unlike normal ERC20 token transfers, the destination address is normally not shown obviously when signing as this is a non-transfer function call.
Recommendation	Consider why there is a need for the `depositTo` function, and if the risks to users outweigh the benefit, consider removing the `depositTo` function.
Resolution	ACKNOWLEDGED The client has not made any change due to other contracts such as rules using this feature.

Issue #04	**setRuleFee is checking against PERFORMANCE_FEE_CAP instead of RULE_EXECUTION_FEE_CAP**
Severity	LOW SEVERITY
Description	Although the values of `PERFORMANCE_FEE_CAP` and `RULE_EXECUTION_FEE_CAP` are the same value (500), `setRuleFee` is using the wrong variable to compare for the upper limit of the `ruleFee` to be set. `function setRuleFee(uint256 _ruleFee) external onlyOwner { require(_ruleFee <= PERFORMANCE_FEE_CAP, 'setRuleFee: CAP_EXCEEDED'); ruleFee = _ruleFee; emit SetRuleFee(_ruleFee); }`
Recommendation	Use `RULE_EXECUTION_FEE_CAP` for the `require` statement.
Resolution	RESOLVED The correct variable is now used.

Issue #05	**Lack of rule and action contract validation**
Severity	LOW SEVERITY
Description	An invalid address for the rule's condition and action contract can be added without any basic sanity checks. Such invalid addresses will result in the rule being unable to be executed.
Recommendation	A sanity check can be used to call functions that would definitely be included in the Rule and Action validation. One simple way would be to ensure that both contracts include a function that returns `true`, and to call that contract in `addRule`.
Resolution	RESOLVED Sanity checks for rules and actions have been added.

Issue #06	**Lack of sanity check for `_stakeToken` and `_strategy` when adding a new pool**
Severity	LOW SEVERITY
Description	An invalid address for a pool's `stakeToken` and `strategy` can be added without any basic sanity checks. Such invalid addresses will result in the pool being unable to function properly.

Recommendation	Consider adding the following sanity checks for `stakeToken` and `strategy`.
	
	`_stakeToken.balanceOf(address(this));`
	`_strategy.totalStakeTokens();`

Resolution	RESOLVED `stakeToken` is now obtained from the strategy.

Issue #07	**Reentrancy guard is not done on `withdrawFrom`**
Severity	LOW SEVERITY
Description	As there is a function (`executeRule`) without reentrancy guard, it might be possible to for reentrancy to occur if the `withdrawFrom` performs an action with some callback behavior to an external contract.
Recommendation	The `nonReentrant` modifier from the external deposit and withdraw related functions should be moved to the internal `_deposit` and `_withdrawFrom` functions instead.
Resolution	RESOLVED The `nonReentrant` modifier has been moved to the internal functions.

Issue #08	Parameter array length checks are not done in addMany
Severity	INFORMATIONAL
Description	In addMany, the length of stakeTokens is used to add both stakeToken and strategies, but it does not check that the length of both arrays are equal. `function addMany(IERC20[] calldata _stakeTokens, IYieldWolfStrategy[] calldata _strategies) external onlyOperator { for (uint256 i; i < _stakeTokens.length; i++) { add(_stakeTokens[i], _strategies[i]); } }`
Recommendation	Check that `_stakeTokens.length == _strategies.length`.
Resolution	RESOLVED The _stakeTokens array have been removed.

Issue #09	Lack of cap for number of rules for each account in a pool
Severity	INFORMATIONAL
Description	Although the rules are not looped through, and only accessed directly using the array index, a lack of cap for rules of an account in a pool allows for an array to grow up to 2^256-1 elements.
Recommendation	Consider adding a maximum limit to the number of rules that can be added for an account in a pool.
Resolution	RESOLVED There is now a maximum cap of 50 rules per pool per user.

Issue #10	**executeRule should be limited to EOA only to reduce the attack surface**
Severity	INFORMATIONAL
Description	As added rules can be customize to the user's liking, there is a risk where an insecure rule could be exploited if called by a contract. One such example would be price manipulation by using a flashloan to cause a temporary massive price movement in the same transaction.
Recommendation	Consider using the following modifier to prevent contract interaction with the `executeRule` function. `modifier onlyEndUser() { require(!isContract(msg.sender) && tx.origin == msg.sender); _; }`
Resolution	RESOLVED The suggested modifier has been added to limit `executeRule` to only EOA.

2.2 AutoCompoundStrategy

There will be an AutoCompoundStrategy contract for each pool in the vault which will be receiving the deposits and staking them into the underlying farm to earn yield. Earned yield can is harvested and then swapped into the staking token which will then be deposited back into the underlying farm, increasing the total balance over time.

ACMiniChef

This contract extends the AutoCompoundStrategy with MiniChef type contracts as the underlying farm.

ACMasterChefWithStaking

This contract extends the AutoCompoundStrategy with MasterChef native staking type contracts as the underlying farm, such as Pancakeswap and Apeswap.

ACMasterChefWithRef

This contract extends the AutoCompoundStrategy with Masterchef with referral type contracts as the underlying farm.

ACMasterChef

This contract extends the AutoCompoundStrategy with the standard MasterChef contracts as the underlying farm.

2.2.1 Privileged Roles and actions

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

- `initialize (can only be called once)`
- `deposit`
- `withdraw`
- `earn`
- `setSwapRouter`
- `setSwapPath`
- `setExtraEarnTokens`

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

- `operator`
- `pause`
- `unpause`
- `emergencyWithdraw`

2.2.2 Issues & Recommendations

Issue #11	**earnAmount might not equate to the same token amount harvested, but balance of the strategy contract**
Severity	HIGH SEVERITY

Description	earnAmount is calculated as earnToken.balanceOf(address(this)); after tokenToEarn is called. This is inaccurate as it can include earnToken balances that are not earned part of the harvest done on the underlying Masterchef.
	
	`uint256 earnAmount = earnToken.balanceOf(address(this));`
	`if (earnAmount == 0) {`
	`return 0;`
	`}`
	
	`(earnAmount, bountyReward) = _distributeFees(earnAmount, _bountyHunter);`
	
	In specific cases like the stake and earn token being the same (e.g. stake Cake for Cake), calling emergencyWithdraw followed by unpause and then earn would result in the _distributeFees calculated based on the entire staking balance, instead of only the harvested amount.

Recommendation	It is recommended to use the delta between after and before harvest to calculate the difference after a harvest is done.
	
	`uint256 before = earnToken.balanceOf(address(this));`
	`// harvest earn tokens`
	`_farmWithdraw(0);`
	`[...]`
	`uint256 earnAmount = earnToken.balanceOf(address(this)) - before;`
	
	`(earnAmount, bountyReward) = _distributeFees(earnAmount, _bountyHunter);`
	
	The _safeSwap using the earnedAmount should continue to use the balance of the contract.

Resolution	RESOLVED Resolved by using the differential between after and before the harvest for distributing rewards.

Issue #12	**Multiple `emergencyWithdraw` calls on underlying farms with deposit/ withdraw fees, or uses a fee on transfer token**
Severity	HIGH SEVERITY
Description	If an underlying farm has deposit/withdraw fees, or uses a fee on transfer token, `emergencyWithdraw` followed by deposit can be repeated, to force withdrawals and deposits multiple times, reducing the total staked balance each time.
Recommendation	`emergencyWithdraw` should only be allowed to be called once, and set a flag to prevent it from being called again, or `unpause` from being called. This flag can also serve as a conditional to prevent `_farmWithdraw(_withdrawAmount);` from being called in `withdraw`. Such a strategy that has been called `emergencyWithdraw` should not be used for further deposits, and users should withdraw funds from the strategy.
Resolution	RESOLVED `emergencyWithdraw` can only be called once, and `unpause` cannot be called after `emergencyWithdraw` has been called.

Issue #13	`totalValueLockedNative` uses LP underlying balances to calculate the pricing
Severity	HIGH SEVERITY
Description	As the `totalValueLockedNative` `view` function uses the balances of the underlying tokens in the liquidity pool contract, it can be easily manipulated by methods such as a flashloan followed by making a huge swap to temporarily change the price. The price can also be manipulated without the use of flashloans if an attacker has sufficient resources, or if the trading pair has low liquidity.
Recommendation	The values returned by `totalValueLockedNative` should never be used for determining the price. Instead, other solutions such as a secure trusted price oracle should be used to prevent price manipulation. Consider removing the function and other related functions that derive the price directly from the liquidity pool contract, as it can be deleted.
Resolution	RESOLVED `totalValueLockedNative` has been removed.

Issue #14	**`setExtraEarnTokens` has no maximum length for `extraEarnTokens`, which can result in denial of service for the `earn` function**
Severity	HIGH SEVERITY
Description	When `extraEarnTokens` is set, it takes an array supplied by the owner, which does not have any length restriction. As `extraEarnTokens` is iterated through in the `earn` function, calling `tokenToEarn` on each of the token addresses, if the array is too long, it can result in the function running out of gas if it exceeds the block gas limit. In such a case, the `earn` function will always revert and never be able to successfully complete. This results in the compound functionality of the strategy to be nonfunctional, defeating the purpose of the vault. Users will also not be able to deposit and withdraw normally as long as earn is called and exceeds the gas limit, leaving insufficient gas for the remaining code to execute. To withdraw, the user would be required to call `emergencyWithdraw`.
Recommendation	Consider adding a maximum length check when setting `extraEarnTokens`.
Resolution	RESOLVED There is a cap of 5 extra `earn` tokens.

Issue #15	`SwapRouter` can be modified by the owner
Severity	HIGH SEVERITY
Description	If the `SwapRouter` is changed to a malicious contract, it would be possible for the owner to steal the tokens in the strategy when external calls such as `safeIncreaseAllowance` are made to the `SwapRouter`. Using a combination of issues such as the lack of token address validation in `tokenToEarn`, as well as the fact that unpause does not automatically deposit all the stake tokens in the contract into the underlying farm, the following would be possible to steal all stake tokens in the strategy. 1. Owner adds stake token address into `extraEarnTokens`. 2. Owner calls `emergencyWithdraw` and all stake tokens are transferred to the strategy from the farm. 3. Owner sets the `SwapRouter` address to a malicious contract. 4. Owner calls unpause, but stake tokens are still in the strategy contract. 5. Owner calls `earn`. `tokenToEarn` will call `_safeSwap` with the stake token address, which will `safeIncreaseAllowance` of the stake token by the entire staked balance to the malicious contract. 6. The `swapExactTokensForTokensSupportingFeeOnTransferTokens` function in the malicious contract can call a `transferFrom` to steal the tokens.
Recommendation	Consider removing the `setSwapRouter` function.
Resolution	RESOLVED `setSwapRouter` has been removed and `setSwapRouterEnabled` will be used instead. If this is set to `true`, the `swapRouter` will be used. Otherwise, the `liquidityRouter` will be used.

Issue #16	Lack of validation of `extraEarnTokens` in `setExtraEarnTokens`
Severity	MEDIUM SEVERITY
Description	If an invalid token address (e.g. non contract address) is added in `extraEarnTokens`, it would cause the `tokenToEarn` function call to revert. Normal deposits and withdrawals would still work due to the `try/catch` on the `tryEarn` function, but the earn function in the strategy would not work.

Recommendation	Before setting `extraEarnTokens`, loop through the array to do a simple sanity check such as the following to ensure that all token addresses are valid.
	`function setExtraEarnTokens(address[] calldata _extraEarnTokens) external virtual onlyOwner {`
	`  for (uint256 i =0; i < _extraEarnTokens.length; i++ {`
	`  IERC20(_extraEarnTokens[i]).balanceOf(address(this));`
	`  }`
	`  extraEarnTokens = _extraEarnTokens;`
	`}`

Resolution	RESOLVED The sanity check has been added.

Issue #17	`farm()` does not check if contract is paused
Severity	MEDIUM SEVERITY
Description	When the contract is paused, `farm` can still be called to force depositing into the underlying farm.
Recommendation	Consider adding the `whenNotPaused` modifier to the `farm` function.
Resolution	RESOLVED The recommended modifier has been added.

Issue #18	`tokenToEarn` lacks sufficient token validation
Severity	LOW SEVERITY
Description	The tokenToEarn has a check that the token to swap is not the `earnToken`: `require(_token != address(earnToken), 'tokenToEarn: NOT_VALID');` However, there are other tokens that should not be swapped too: `stakeToken`, `token0` and `token1`. `stakeToken` would not be able to swap if it is an LP token and fail, while it is unnecessary to swap `token0` and `token1` to the `earnToken` as the earned token would later be swapped to those tokens.
Recommendation	Consider adding additional checks to prevent `stakeToken`, `token0` and `token1` to be swapped. These additional checks can also be done at the function that `setExtraEarnTokens` instead.
Resolution	RESOLVED There is now a check to ensure that `stakeToken` cannot be used for swapping in `tokenToEarn`.

Issue #19	`tokenToEarn` reverts if `earnToken` is a token attempted to be swapped
Severity	LOW SEVERITY
Description	The check to ensure that the token to be swapped is not the `earnToken` is done using a require statement, which would revert the entire earn function call. This would prevent the `earn` function from working if `extraEarnTokens` contains the `earnToken`.
Recommendation	Consider using an `if` statement and returning early if it is a token that should not be swapped.
Resolution	RESOLVED The `if` statement is now used instead of the `require` statement.

Issue #20	**`tokenToEarn` can fail if the token does not have a native token pair, or the liquidity pool has insufficient liquidity for a successful swap**
Severity	LOW SEVERITY
Description	If a provided token does not have a native pair, the route provided for the swap will not exist and cause `tokenToEarn` to fail.
Recommendation	Consider removing the loop of `extraEarnTokens` from `earn`, and putting it in a separate function. Even if it fails, `earn` will not be affected.
Resolution	RESOLVED Resolved by introducing a try-catch. Furthermore, a full integration test will be run before adding any pool to ensure all pairs exists and all operations succeed. No pool will be added if the integration test doesn't succeed.

Issue #21	**`block.timestamp` can be used as the deadline for swaps and adding liquidity**
Severity	INFORMATIONAL
Description	There is no need to use `block.timestamp` + 200 as the deadline for the router function calls, as the check is the following: `require(deadline >= block.timestamp, 'UniswapV2Router: EXPIRED');` As the swap or add liquidity call is happening in the same transaction, the deadline would be equals to the block.timestamp.
Recommendation	Consider using `block.timestamp` for the deadline parameter for adding liquidity and swaps.
Resolution	RESOLVED The unnecessary addition has been removed.

Issue #22	Swap and liquidity router can be different addresses
Severity	INFORMATIONAL
Description	The swap and liquidity router can be separate addresses. In a normal case, these two should be the same as you would usually want to swap and provide liquidity on the same router.
Recommendation	Consider merging the two router variables into one, which will be used for both adding liquidity and swaps.
Resolution	ACKNOWLEDGED The client has acknowledged that this is intended. The staked pair LP may not provide the best liquidity, so the swap LP on another exchange can be used to improve liquidity for swaps.

Issue #23	`token0` and `token1` should be obtained from the LP contract if stake token is an LP
Severity	INFORMATIONAL
Description	If the stake token is an LP, the `token0` and `token1` are manually set in the constructor. This could result in the contract not functioning properly if there is any human error and the `token0` or `token1` are not actually the underlying tokens of the staked LP token.
Recommendation	Consider removing the `token0` and `token1` parameters in the constructor, instead adding an `isLP` bool flag to determine if the `token0` and `token1` are obtained from the factory, or set as the zero address if the stake token is not an LP token.
Resolution	RESOLVED `token0` and `token1` is obtained from the LP contract.

Issue #24	`setReferrer` allows setting strategy's referrer in the underlying farm
Severity	INFORMATIONAL
Description	`setReferrer` allows setting strategy's referrer in the underlying farm, allowing for that address to earn referrer rewards.
Recommendation	Consider documenting how the referrer fees will be used.
Resolution	ACKNOWLEDGED Proper documentation will be done before the launch.

Issue #25	`earn` may fail if the underlying farm prevents harvesting with 0 as the withdraw amount
Severity	INFORMATIONAL
Description	In general, Masterchef forks do not prevent calling the `withdraw` function with 0 as the parameter, but there have been some farms that prevent harvesting with withdraw as 0. For such a farm, the `_farmWithdraw` would fail, causing `earn` to not work.
Recommendation	Ensure that due diligence is done on an underlying farm before implementing and deploying a strategy for it.
Resolution	RESOLVED A change has been added so the `earn` function calls an internal and virtual `_farmHarvest()`. This will make the abstract contract easier to extend if a future farm requires a different type of harvesting method. On ACMiniChef (used by Sushi and ApeSwap on Polygon), the `_farmHarvest` was added to call the harvest method on their contract as they do not harvest with `deposit(0)`. Furthermore, the YieldWolf team will run a complete integration test on a fork of Polygon before adding a pool. This test includes depositing with multiple users, advancing the blockchain certain number of blocks (or seconds in a few cases), compounding rewards, and then withdrawing and depositing again.

2.3 NativeValueLowerThanCondition

This contract is a simple example Rule contract which checks if the criteria is met before the rule can be executed. The provided contract should not be used in production. Users are to do their own due diligence on the Rule contract they use when adding a rule to their staking position in YieldWolf.

2.3.1 Issues & Recommendations

Issue #26	Rule is vulnerable to price manipulation attacks
Severity	HIGH SEVERITY
Description	Uses `totalValueLockedNative`, which gets the current price based on the underlying LP's `token0` and `token1` balance. It will be vulnerable to flash loan attacks which can manipulate the price.
Recommendation	Do not use this rule in production. Also, this rule should not be provided as an example to users.
Resolution	RESOLVED The example rule has been replaced.

Issue #27	Improper calculation of `userNativeValue`
Severity	HIGH SEVERITY

Description	The `userNativeValue` is calculated as such:
	`uint256 userNativeValue = (strategy.sharesTotal() * totalNativeValue) / stakedTokens;`
	However, this is incorrect since `stakedTokens` is the user's staked token amount, not share amount. It should be the following instead:
	`uint256 userNativeValue =  totalNativeValue * stakedTokens / strategy.totalStakeTokens();`

Recommendation	Correct the formula used for calculation of the `userNativeValue`.
Resolution	RESOLVED The example rule has been replaced.


PALADIN
BLOCKCHAIN SECURITY