



Pangolin Staking Positions Audit Report

Aug 30, 2022





Table of Contents

Summary	2
Overview	3
Issues	4
WP-H1: Add liquidity to the PNG-WAVAX pool in compoundToPoolZero() or compound() with msg.value is not handled properly	4
WP-M2: Insufficient slippage control in _addLiquidity() with maxPairAmount	8
WP-N3: Misleading variable names	11
WP-G4: Existence of pool.rewarder should be checked before the IReward.onReward() call for consistency and gas saving	14
WP-G5: Immutable variables should not be copied to memory	19
Appendix	22
Disclaimer	23



Summary

This report has been prepared for Pangolin Staking Positions Audit Report smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.



Overview

Project Summary

Project Name	Pangolin Staking Positions
Codebase	https://github.com/pangolindex/exchange-contracts
Commit	7c01b8f562e2095a36a3689cd1e587ef630b242c
Language	Solidity

Audit Summary

Delivery Date	Aug 30, 2022
Audit Methodology	Static Analysis, Manual Review
Total Issues	5



WP-H1: Add liquidity to the PNG-WAVAX pool in compoundToPoolZero() or compound() with msg.value is not handled properly

High

Issue Description

<https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L641-L687>

```
641 function _addLiquidity(  
642     Pool storage pool,  
643     uint256 rewardAmount,  
644     uint256 maxPairAmount  
645 ) private returns (uint256 poolTokenAmount) {  
646     address poolToken = pool.tokenOrRecipient;  
647     address rewardPair = pool.rewardPair;  
648  
649     // Get token amounts from the pool.  
650     (uint256 reserve0, uint256 reserve1, ) =  
        IPangolinPair(poolToken).getReserves();  
651  
652     // Get the reward token's pair's amount from the reserves.  
653     ERC20 tmpRewardsToken = rewardsToken;  
654     uint256 pairAmount = address(tmpRewardsToken) < rewardPair  
655         ? (reserve1 * rewardAmount) / reserve0  
656         : (reserve0 * rewardAmount) / reserve1;  
657  
658     // Ensure slippage is not above the limit.  
659     if (pairAmount > maxPairAmount) revert HighSlippage();  
660  
661     // Non-zero message value signals desire to pay with native token.  
662     if (msg.value > 0) {  
663         // Ensure reward pair is native token.  
664         if (rewardPair != wrappedNativeToken) revert InvalidToken();  
665  
666         // Ensure consistent slippage control.  
667         if (msg.value != maxPairAmount) revert InvalidAmount();
```



```
668
669     // Wrap the native token.
670     IWAVAX(rewardPair).deposit{ value: pairAmount }();
671
672     // Refund user.
673     unchecked {
674         uint256 refundAmount = msg.value - pairAmount;
675         if (refundAmount != 0) SafeTransferLib.safeTransferETH(msg.sender,
refundAmount);
676     }
677 }
678
679     // Transfer reward pair tokens from the user to the pair contract.
680     ERC20(rewardPair).safeTransferFrom(msg.sender, poolToken, pairAmount);
681
682     // Transfer reward tokens from the contract to the pair contract.
683     tmpRewardsToken.safeTransfer(poolToken, rewardAmount);
684
685     // Mint Liquidity tokens to the PangoChef and return the amount minted.
686     poolTokenAmount = IPangolinPair(poolToken).mint(address(this));
687 }
```

When the user `compoundToPoolZero()` or `compound()`, a certain amount of native tokens can be sent as `msg.value` to be paired with the `rewardsToken` (`PNG`) and add liquidity to the `PNG-WAVAX` pool.

The user can also choose not to pay with the native token, but with `WAVAX`. The contract will then pull the funds from the user directly to the `PNG-WAVAX` pool (L680).

However, the current implementation is not handling the native token payment (`msg.value`) correctly: the native tokens will be wrapped (L670), but the wrapped native tokens will not be sent to the `poolToken` contract.

Instead, it will continue to pull funds from the user's wallet to the `PNG-WAVAX` pool (L680).

As a result, the users who pay with `msg.value` will be paying double the amount of native tokens.



Recommendation

Change to:

```
641 function _addLiquidity(  
642     Pool storage pool,  
643     uint256 rewardAmount,  
644     uint256 maxPairAmount  
645 ) private returns (uint256 poolTokenAmount) {  
646     address poolToken = pool.tokenOrRecipient;  
647     address rewardPair = pool.rewardPair;  
648  
649     // Get token amounts from the pool.  
650     (uint256 reserve0, uint256 reserve1, ) =  
        IPangolinPair(poolToken).getReserves();  
651  
652     // Get the reward token's pair's amount from the reserves.  
653     ERC20 tmpRewardsToken = rewardsToken;  
654     uint256 pairAmount = address(tmpRewardsToken) < rewardPair  
655         ? (reserve1 * rewardAmount) / reserve0  
656         : (reserve0 * rewardAmount) / reserve1;  
657  
658     // Ensure slippage is not above the limit.  
659     if (pairAmount > maxPairAmount) revert HighSlippage();  
660  
661     // Non-zero message value signals desire to pay with native token.  
662     if (msg.value > 0) {  
663         // Ensure reward pair is native token.  
664         if (rewardPair != wrappedNativeToken) revert InvalidToken();  
665  
666         // Ensure consistent slippage control.  
667         if (msg.value != maxPairAmount) revert InvalidAmount();  
668  
669         // Wrap the native token.  
670         IWAVAX(rewardPair).deposit{ value: pairAmount }();  
671         // Transfer reward pair tokens from this contract to the pair contract.  
672         ERC20(rewardPair).safeTransfer(poolToken, pairAmount);  
673  
674         // Refund user.  
675         unchecked {  
676             uint256 refundAmount = msg.value - pairAmount;  
677             if (refundAmount != 0) SafeTransferLib.safeTransferETH(msg.sender,  
                refundAmount);
```



```
678     }
679   } else {
680     // Transfer reward pair tokens from the user to the pair contract.
681     ERC20(rewardPair).safeTransferFrom(msg.sender, poolToken, pairAmount);
682   }
683
684   // Transfer reward tokens from the contract to the pair contract.
685   tmpRewardsToken.safeTransfer(poolToken, rewardAmount);
686
687   // Mint liquidity tokens to the PangoChef and return the amount minted.
688   poolTokenAmount = IPangolinPair(poolToken).mint(address(this));
689 }
```

Status

✓ Fixed



WP-M2: Insufficient slippage control in `_addLiquidity()` with `maxPairAmount`

Medium

Issue Description

When the user `compoundToPoolZero()` or `compound()`, a certain amount of other tokens (native or ERC20 tokens) can be paired with the `rewardsToken` (`PNG`) and add liquidity to the `PNG` -paired pool.

According to the comments, the `maxPairAmount` parameter is used for `slippage control` .

While it does put a upper limit to the amount of paired tokens to be spent for adding liquidity, which is useful to some extend, to prevent overspending, but we believe it's insufficient to prevent MEV.

<https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L641-L687>

```
641 function _addLiquidity(  
642     Pool storage pool,  
643     uint256 rewardAmount,  
644     uint256 maxPairAmount  
645 ) private returns (uint256 poolTokenAmount) {  
646     address poolToken = pool.tokenOrRecipient;  
647     address rewardPair = pool.rewardPair;  
648  
649     // Get token amounts from the pool.  
650     (uint256 reserve0, uint256 reserve1, ) =  
        IPangolinPair(poolToken).getReserves();  
651  
652     // Get the reward token's pair's amount from the reserves.  
653     ERC20 tmpRewardsToken = rewardsToken;  
654     uint256 pairAmount = address(tmpRewardsToken) < rewardPair  
655         ? (reserve1 * rewardAmount) / reserve0  
656         : (reserve0 * rewardAmount) / reserve1;  
657  
658     // Ensure slippage is not above the limit.
```



```
659     if (pairAmount > maxPairAmount) revert HighSlippage();
660
661     // Non-zero message value signals desire to pay with native token.
662     if (msg.value > 0) {
663         // Ensure reward pair is native token.
664         if (rewardPair != wrappedNativeToken) revert InvalidToken();
665
666         // Ensure consistent slippage control.
667         if (msg.value != maxPairAmount) revert InvalidAmount();
668
669         // Wrap the native token.
670         IWAVAX(rewardPair).deposit{ value: pairAmount }();
671
672         // Refund user.
673         unchecked {
674             uint256 refundAmount = msg.value - pairAmount;
675             if (refundAmount != 0) SafeTransferLib.safeTransferETH(msg.sender,
refundAmount);
676         }
677     }
678
679     // Transfer reward pair tokens from the user to the pair contract.
680     ERC20(rewardPair).safeTransferFrom(msg.sender, poolToken, pairAmount);
681
682     // Transfer reward tokens from the contract to the pair contract.
683     tmpRewardsToken.safeTransfer(poolToken, rewardAmount);
684
685     // Mint liquidity tokens to the PangoChef and return the amount minted.
686     poolTokenAmount = IPangolinPair(poolToken).mint(address(this));
687 }
```



PoC

Given:

- `poolToken` : `PNG-USDC` ;
 - Current market price of `PNG` is: `10 USDC` ;
 - Current `PNG-USDC` pool reserves: `10 PNG` and `100 USDC` . (We use a lower liquidity as an example for easier manipulation. The attack vector works for pools with regular liquidity as well.)
1. Alice called `stake()` to compounding `PNG-USDC` with `10k PNG` and `100k USDC` as `maxPairAmount` ;
 2. Attacker frontrun Alice's `stake()` transaction and skewed the price of `PNG` in `PNG-USDC` pool to `1 USDC` ;
 3. By the time Alice's `stake()` was minted, only `10k USDC` was added (at `1 USDC per PNG`);
 4. Attacker backrun Alice's `stake()` transaction and bought most of the newly added `PNG` tokens at a huge discount.

Recommendation

Consider adding a new parameter `minPairAmount` for slippage control.



WP-N3: Misleading varibale names

Issue Description

1. “user” sounds like the user’s address, while it’s actually a struct that storing the user’s information in the pool:

<https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L586-L628>

```
586 function _harvestWithoutReset(uint256 poolId) private returns (uint256 reward) {
587     // Create a storage pointer for the pool and the user.
588     Pool storage pool = pools[poolId];
589     User storage user = pool.users[msg.sender];
590
591     // Ensure pool is ERC20 type.
592     _onlyERC20Pool(pool);
593
594     // Update pool summations that govern the reward distribution from pool to
    users.
595     _updateRewardSummations(poolId, pool);
596
597     // Pool zero should instead use `compound()`.
598     if (poolId == 0) revert InvalidType();
599
600     // Increment Lock count on pool zero if this pool was not already locking it.
601     _incrementLockOnPoolZero(user);
602
603     // Get the rewards accrued by the user, then delete the user stash.
604     reward = _userPendingRewards(poolId, pool, user);
605     user.stashedRewards = 0;
606
607     // Ensure there are sufficient rewards to use in compounding.
608     if (reward == 0) revert NoEffect();
609
610     // Increment the previousValues to not reset the staking duration. In the
    proofs,
611     // previousValues was regarding combining positions, however we are not
    combining positions
612     // here. Consider this trick as combining with a null position. This allows us
    to not reset
```



```
613     // the staking duration but exclude any rewards before block time.
614     uint256 userBalance = user.valueVariables.balance;
615     user.previousValues += uint152(userBalance * (block.timestamp -
        user.lastUpdate));
616
617     // Snapshot the LastUpdate and summations.
618     _snapshotRewardSummations(pool, user);
619
620     // Emit the harvest event, even though it will not be transferred to the user.
621     emit Withdrawn(poolId, msg.sender, 0, reward);
622
623     // Get extra rewards from rewarder.
624     IRewarder rewarder = pool.rewarder;
625     if (address(rewarder) != address(0)) {
626         rewarder.onReward(poolId, msg.sender, msg.sender, reward, userBalance);
627     }
628 }
```

Recommendation

Consider renaming to `poolUser` .

2. “pool.rewardPair” sounds like the address of the pair, while it’s actually the address of the token that paired the ”rewardToken”

<https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L88-L105>

```
88     struct Pool {
89         // The address of the token when poolType is ERC_20, or the recipient address
        when poolType
90         // is RELAYER_POOL.
91         address tokenOrRecipient;
92         // The type of the pool, which determines which actions can be performed on
        it.
93         PoolType poolType;
94         // An external contract that distributes additional rewards.
95         IRewarder rewarder;
96         // The address that is paired with PNG. It is zero address if the pool token
        is not a
```



```
97      // liquidity pool token, or if the liquidity pool do not have PNG as one of
the reserves.
98      address rewardPair;
99      // Two variables that specify the total shares (i.e.: "value") in the pool.
100     ValueVariables valueVariables;
101     // Summations incremented on every action on the pool.
102     RewardSummations rewardSummationsStored;
103     // The mapping from addresses of the users of the pool to their properties.
104     mapping(address => User) users;
105 }
```

<https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L641-L647>

```
641     function _addLiquidity(
642         Pool storage pool,
643         uint256 rewardAmount,
644         uint256 maxPairAmount
645     ) private returns (uint256 poolTokenAmount) {
646         address poolToken = pool.tokenOrRecipient;
647         address rewardPair = pool.rewardPair;
```

Recommendation

Consider renaming to `rewardPairedToken` .

Status

 Acknowledged



WP-G4: Existence of pool.rewarder should be checked before the IReward.onReward() call for consistency and gas saving

Gas

Issue Description

The rewarder may not exist and other methods like `_stake()` (L503), `_withdraw()` (L574), `_harvestWithoutReset()` (L625) will check the existence of rewarder before calling `IReward.onReward()`.

<https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L513-L578>

```
513 function _withdraw(uint256 poolId, uint256 amount) private {
514     // Create a storage pointer for the pool and the user.
515     Pool storage pool = pools[poolId];
516     User storage user = pool.users[msg.sender];
517
518     // Ensure pool is ERC20 type.
519     _onlyERC20Pool(pool);
520
521     // Update pool summations that govern the reward distribution from pool to
    users.
522     _updateRewardSummations(poolId, pool);
523
524     // Ensure pool zero is not locked.
525     // Decrement lock count on pool zero if this pool was locking it.
526     _decrementLockOnPoolZero(poolId, user);
527
528     // Get position balance and ensure sufficient balance exists.
529     ValueVariables storage userValueVariables = user.valueVariables;
530     uint256 oldBalance = userValueVariables.balance;
531     if (amount > oldBalance) revert InsufficientBalance();
532
533     // Before everything else, get the rewards accrued by the user, then delete
    the user stash.
534     uint256 reward = _userPendingRewards(poolId, pool, user);
535     user.stashedRewards = 0;
```



```
536
537     // Ensure we are either withdrawing something or claiming rewards.
538     if (amount == 0 && reward == 0) revert NoEffect();
539
540     uint256 remaining;
541     unchecked {
542         // Get the remaining balance in the position.
543         remaining = oldBalance - amount;
544
545         // Decrement the withdrawn amount from totalStaked.
546         ValueVariables storage poolValueVariables = pool.valueVariables;
547         poolValueVariables.balance -= uint104(amount);
548
549         // Update sumOfEntryTimes.
550         uint256 newEntryTimes = block.timestamp * remaining;
551         poolValueVariables.sumOfEntryTimes = uint152(
552             poolValueVariables.sumOfEntryTimes +
553             newEntryTimes -
554             userValueVariables.sumOfEntryTimes
555         );
556
557         // Decrement the withdrawn amount from user balance, and update the user
558         entry times.
559         userValueVariables.balance = uint104(remaining);
560         userValueVariables.sumOfEntryTimes = uint152(newEntryTimes);
561     }
562
563     // Reset the previous values, as we have restarted the staking duration.
564     user.previousValues = 0;
565
566     // Snapshot the lastUpdate and summations.
567     _snapshotRewardSummations(pool, user);
568
569     // Transfer withdrawn tokens.
570     if (reward != 0) rewardsToken.safeTransfer(msg.sender, reward);
571     if (amount != 0) ERC20(pool.tokenOrRecipient).safeTransfer(msg.sender,
572 amount);
573     emit Withdrawn(poolId, msg.sender, amount, reward);
574
575     // Get extra rewards from rewarder if it is not an emergency exit.
576     IRewarder rewarder = pool.rewarder;
577     if (address(rewarder) != address(0)) {
578         rewarder.onReward(poolId, msg.sender, msg.sender, reward, remaining);
579     }
```




```
577     }  
578 }
```

By contrast, `_emergencyExit()` will call `pool.rewarder` without checking if `address(pool.rewarder) != address(0)` .

We believe that's inconsistent and will waste some gas if `rewarder == address(0)` .

<https://github.com/pangolindex/exchange-contracts/blob/6739cd2ed98903c39f8173eff7565c56e4d97456/contracts/staking-positions/PangoChef.sol#L694-L756>

```
694  function _emergencyExit(uint256 poolId, bool withdrawStake) private {  
695      // Create storage pointers for the pool and the user.  
696      Pool storage pool = pools[poolId];  
697      User storage user = pool.users[msg.sender];  
698  
699      // Ensure pool is ERC20 type.  
700      _onlyERC20Pool(pool);  
701  
702      // Decrement lock count on pool zero if this pool was locking it.  
703      _decrementLockOnPoolZero(poolId, user);  
704  
705      // Create storage pointers for the value variables.  
706      ValueVariables storage poolValueVariables = pool.valueVariables;  
707      ValueVariables storage userValueVariables = user.valueVariables;  
708  
709      // Decrement the state variables pertaining to total value calculation.  
710      uint104 balance = userValueVariables.balance;  
711      if (balance == 0) revert NoEffect();  
712      unchecked {  
713          poolValueVariables.balance -= balance;  
714          poolValueVariables.sumOfEntryTimes -= userValueVariables.sumOfEntryTimes;  
715      }  
716  
717      // Simply delete the user information.  
718      delete pools[poolId].users[msg.sender];  
719  
720      // Transfer stake from contract to user and emit the associated event.  
721      if (withdrawStake) {
```




```
722     ERC20(pool.tokenOrRecipient).safeTransfer(msg.sender, balance);
723     emit Withdrawn(poolId, msg.sender, balance, 0);
724     // Still try withdrawing, but do a non-reverting low-level call.
725     } else {
726         (bool success, bytes memory returndata) = pool.tokenOrRecipient.call(
727             abi.encodeWithSelector(ERC20.transfer.selector, msg.sender, balance)
728         );
729         if (success && returndata.length > 0 && abi.decode(returndata, (bool))) {
730             emit Withdrawn(poolId, msg.sender, balance, 0);
731         }
732     }
733
734     {
735         // Do a low-level call for rewarder. If external function reverts, only
736         the external
737         // contract reverts. To prevent DOS, this function (_emergencyExit) must
738         never revert
739         // unless `balance == 0`. This can still return true if rewarder is not a
740         contract.
741         (bool success, ) = address(pool.rewarder).call(
742             abi.encodeWithSelector(
743                 IRewarder.onReward.selector,
744                 poolId,
745                 msg.sender,
746                 msg.sender,
747                 0,
748                 0
749             )
750         );
751
752         // Record last failed Rewarder calls. This can be used for slashing
753         rewards by a
754         // non-malicious Rewarder just in case it reverts due to some bug. If
755         rewarder is
756         // correctly written, this statement should never execute. We also do not
757         care if
758         // `success` is `true` due to rewarder not being a contract. A
759         non-contract rewarder
760         // only means that it is unset. So it does not matter if we record or not.
761         if (!success) lastTimeRewarderCallFailed[poolId][msg.sender] =
762             block.timestamp;
763     }
764 }
```



Recommendation

Consider checking if `address(pool.rewarder) != address(0)` before doing the low-level call for rewarder.

Status

 Acknowledged



WP-G5: Immutable variables should not be copied to memory

Gas

Issue Description

<https://docs.soliditylang.org/en/v0.8.16/contracts.html?highlight=immutable#constant-and-immutable-state-variables>

Immutable variables are evaluated once at construction time and their value is copied to all the places in the code where they are accessed.

Coping immutable variables to memory is a waste of gas. For example:

<https://github.com/pangolinindex/exchange-contracts/blob/328067a5af7bbc67360ca9a0c5bda7c350234371/contracts/staking-positions/PangoChefFunding.sol#L81-L82>

```
81      /** @notice The reward token that is distributed to stakers. */  
82      ERC20 public immutable rewardsToken;
```

<https://github.com/pangolinindex/exchange-contracts/blob/328067a5af7bbc67360ca9a0c5bda7c350234371/contracts/staking-positions/PangoChef.sol#L647-L697>

```
647      function _addLiquidity(  
648          Pool storage pool,  
649          uint256 rewardAmount,  
650          Slippage memory slippage  
651      ) private returns (uint256 poolTokenAmount) {  
652          address poolToken = pool.tokenOrRecipient;  
653          address rewardPair = pool.rewardPair;  
654  
655          // Get token amounts from the pool.  
656          (uint256 reserve0, uint256 reserve1, ) =  
            IPangolinPair(poolToken).getReserves();  
657  
658          // Get the reward token's pair's amount from the reserves.
```



```
659     ERC20 tmpRewardsToken = rewardsToken;
660     uint256 pairAmount = address(tmpRewardsToken) < rewardPair
661         ? (reserve1 * rewardAmount) / reserve0
662         : (reserve0 * rewardAmount) / reserve1;
663
664     // Ensure slippage is not above the limit.
665     if (pairAmount > slippage.maxPairAmount) revert HighSlippage();
666     if (pairAmount < slippage.minPairAmount) revert HighSlippage();
667
668     // Non-zero message value signals desire to pay with native token.
669     if (msg.value > 0) {
670         // Ensure reward pair is native token.
671         if (rewardPair != wrappedNativeToken) revert InvalidToken();
672
673         // Ensure consistent slippage control.
674         if (msg.value != slippage.maxPairAmount) revert InvalidAmount();
675
676         // Wrap the native token.
677         IWAVAX(rewardPair).deposit{ value: pairAmount }();
678
679         // Transfer reward pair tokens from this contract to the pair
contract.
680         ERC20(rewardPair).safeTransfer(poolToken, pairAmount);
681
682         // Refund user.
683         unchecked {
684             uint256 refundAmount = msg.value - pairAmount;
685             if (refundAmount != 0) SafeTransferLib.safeTransferETH(msg.sender,
refundAmount);
686         }
687     } else {
688         // Transfer reward pair tokens from the user to the pair contract.
689         ERC20(rewardPair).safeTransferFrom(msg.sender, poolToken, pairAmount);
690     }
691
692     // Transfer reward tokens from the contract to the pair contract.
693     tmpRewardsToken.safeTransfer(poolToken, rewardAmount);
694
695     // Mint Liquidity tokens to the PangoChef and return the amount minted.
696     poolTokenAmount = IPangolinPair(poolToken).mint(address(this));
697 }
```



Recommendation

Using `rewardsToken` directly and remove `tmpRewardsToken` .

Status

✓ Fixed



Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.



Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.