// HALBORN

Aura Finance - AuraBAL Compounder

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 6th, 2023 - March 20th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/19/2023	Ataberk Yavuzer
0.2	Document Edits	03/20/2023	Ataberk Yavuzer
0.3	Final Draft	03/20/2023	Ataberk Yavuzer
1.0	Remediation Plan	03/20/2023	Ataberk Yavuzer
1.1	Remediation Plan Review	03/20/2023	Grzegorz Trawinski
1.2	Remediation Plan Review	03/20/2023	Piotr Cielas
1.3	Remediation Plan Review	03/21/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com
Grzegorz Trawinski	Halborn	Grzegorz.Trawinski@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Aura Finance is a protocol built on top of the Balancer system to provide maximum incentives to Balancer liquidity providers and BAL stakers (into veBAL) through social aggregation of BAL deposits and Aura's native token.

Aura Finance engaged Halborn to conduct a security audit on their smart contracts beginning on March 6th, 2023 and ending on March 20th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were addressed and accepted by the Aura Finance team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that don't follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart Contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual Assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Dynamic Analysis (foundry)
- Static Analysis(slither, MythX)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. Aura Finance - AuraBAL Compounder Audit Test Scope
   (a) Repository: compounder
   (b) Commit ID: 5296b21a08280afef1d3d31964c65bc9f17d391f
2. In-Scope:
   (a) **compounder/*.sol**

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	1	3	4

LIKELIHOOD

IMPACT

🟨	🟨	🟥 (HAL-01)	🟥	🟪
🟩 (HAL-03)	🟨	🟨	🟥	🟥
🟩	🟩	🟨	🟨	🟥
⬜	🟩	🟩	🟨 (HAL-02)	🟨
⬜ (HAL-07) (HAL-08) (HAL-09)	⬜ (HAL-06)	🟩 (HAL-04) (HAL-05)	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) THE FIRST DEPOSITOR MAY LOSE FUNDS DUE TO ERC4626 ROUNDING ISSUE	High	SOLVED - 03/19/2023
(HAL-02) PENALTY FEE CAN BE BYPASSED	Medium	SOLVED - 03/19/2023
(HAL-03) THE LAST HARVEST OPERATION CAN BE SANDWICHED	Low	RISK ACCEPTED
(HAL-04) DUPLICATE REWARD TOKENS CAN BE ADDED	Low	RISK ACCEPTED
(HAL-05) LACK OF TWO-STEP OWNERSHIP PATTERN	Low	RISK ACCEPTED
(HAL-06) MISTAKENLY SEND ETHERS LOCKED FOREVER IN VAULTS	Informational	SOLVED - 03/19/2023
(HAL-07) FOR LOOP OPTIMIZATIONS	Informational	ACKNOWLEDGED
(HAL-08) REDUNDANT CODE	Informational	ACKNOWLEDGED
(HAL-09) INCORRECT COMMENTS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) THE FIRST DEPOSITOR MAY LOSE FUNDS DUE TO ERC4626 ROUNDING ISSUE - HIGH

Description:

During the initial asset deposit in ERC4626 Vaults, the first liquidity provider can lose funds due to rounding issues.

The risk above is explained in **EIP4626** standard:
> Finally, ERC-4626 Vault implementers should be aware of the need for specific, opposing rounding directions across the different mutable and view methods, as it is considered most secure to favor the Vault itself during calculations over its users:

> If (1) it's calculating how many shares to issue to a user for a certain amount of the underlying tokens they provide or (2) it's determining the amount of the underlying tokens to transfer to them for returning a certain amount of shares, it should **round down**.

> If (1) it's calculating the amount of shares a user has to supply to receive a given amount of the underlying tokens or (2) it's calculating the amount of underlying tokens a user has to provide to receive a certain amount of shares, it should **round up**.

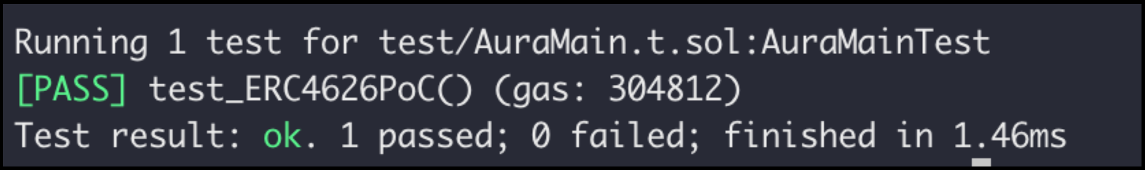
The current GenericUnionVault contract does not implement any rounding validations as discussed above. Thus, the contract is vulnerable to a front-running attack.

In this case, any attacker can front-run the first deposit operation to claim more assets during the redeem() call.

Proof of Concept:

Listing 1: PoC Code - ERC4626 Vulnerability

```solidity
function test_ERC4626PoC() public {
    vm.startPrank(user1);

    uint256 preBalanceUser1 = testToken.balanceOf(user1);
    uint256 preBalanceUser2 = testToken.balanceOf(user2);

    testToken.approve(address(vault), 1);
    vault.mint(1, user1);
    strategyM.stake(20e18);

    vm.stopPrank();

    vm.startPrank(user2);
    testToken.approve(address(vault), 100e18);
    vault.deposit(100e18, user2);
    vm.stopPrank();

    vm.prank(user1);
    vault.redeem(1, user1, user1);

    uint256 afterBalanceUser1 = testToken.balanceOf(user1);

    assertGt(afterBalanceUser1, preBalanceUser1);

    vm.prank(user2);
    vault.redeem(3, user2, user2); //! max redeem returns 4. However, it will fail due to price miscalculation so we will use 3.
    uint256 afterBalanceUser2 = testToken.balanceOf(user2);

    assertLt(afterBalanceUser2, preBalanceUser2);
}
```



```
Running 1 test for test/AuraMain.t.sol:AuraMainTest
[PASS] test_ERC4626PoC() (gas: 304812)
Test result: ok. 1 passed; 0 failed; finished in 1.46ms
```

Code Location:

Listing 2: GenericVault.sol (Lines 114,116)

```
104 function deposit(address _to, uint256 _amount) public
 ↳ notToZeroAddress(_to) returns (uint256 _shares) {
105     require(_amount > 0, "Deposit too small");
106
107     uint256 _before = totalUnderlying();
108     IERC20(underlying).safeTransferFrom(msg.sender, strategy,
 ↳ _amount);
109     IStrategy(strategy).stake(_amount);
110
111     // Issues shares in proportion of deposit to pool amount
112     uint256 shares = 0;
113     if (totalSupply() == 0) {
114         shares = _amount;
115     } else {
116         shares = (_amount * totalSupply()) / _before;
117     }
118     _mint(_to, shares);
119     emit Deposit(msg.sender, _to, _amount);
120     return shares;
121 }
```

Listing 3: GenericVault.sol (Line 136)

```
133 function _withdraw(uint256 _shares) internal returns (uint256
 ↳ _withdrawable) {
134     require(totalSupply() > 0);
135     // Computes the amount withdrawable based on the number of
 ↳ shares sent
136     uint256 amount = (_shares * totalUnderlying()) / totalSupply()
 ↳ ;
137     // Burn the shares before retrieving tokens
138     _burn(msg.sender, _shares);
139     // If user is last to withdraw, harvest before exit
140     if (totalSupply() == 0) {
141         harvest();
142         IStrategy(strategy).withdraw(totalUnderlying());
143         _withdrawable = IERC20(underlying).balanceOf(address(this)
 ↳ );
144     }
145     // Otherwise compute share and unstake
```

```
    else {
        _withdrawable = amount;
        // Substract a small withdrawal fee to prevent users "
↳ timing"
        // the harvests. The fee stays staked and is therefore
        // redistributed to all remaining participants.
        uint256 _penalty = (_withdrawable * withdrawalPenalty) /
↳ FEE_DENOMINATOR;
        _withdrawable = _withdrawable - _penalty;
        IStrategy(strategy).withdraw(_withdrawable);
    }
    return _withdrawable;
}
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

The contract should do an **INITIAL DEPOSIT** for an arbitrary address to prevent this attack from happening. For example, some amounts should be deposited for the zero address initially.

Also, the contract should follow the official **Rounding** recommendations for ERC4626 functions as provided by OpenZeppelin:

- ERC4626.sol

Remediation Plan:

SOLVED: The Aura Finance team solved this issue by implementing a new rounding pattern in their code.

Commit ID: diff-e4193a9b. . . b4d6e0ee04R361

3.2 (HAL-02) PENALTY FEE CAN BE BYPASSED - MEDIUM

Description:

The withdraw penalty fee formula does not take precision into account. Therefore, it is possible to bypass the penalty fee by requesting a withdraw of a small amount of shares.

The penalty fee formula parameters in the default configuration of GenericUnionVault contract are given below:

Listing 4: Penalty Fee Formula - Variables

```
withdrawalPenalty = 100;
FEE_DENOMINATOR = 10000;

_penalty = (_withdrawable * withdrawalPenalty) / FEE_DENOMINATOR;
```

If _withdrawable variable returns less than 100, then _penalty will be always zero.

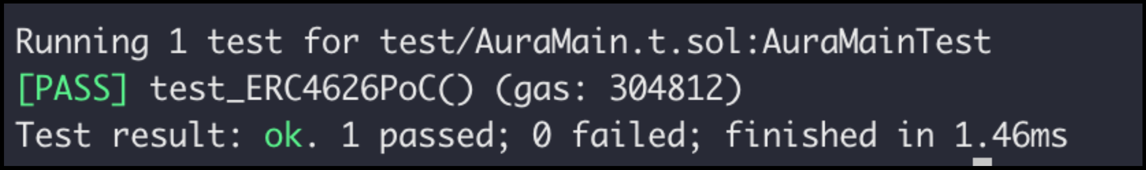
Listing 5: Case 1 - Withdrawable less than 100

```
withdrawalPenalty = 100;
FEE_DENOMINATOR = 10000;
_withdrawable = 97;

_penalty = (_withdrawable * withdrawalPenalty) / FEE_DENOMINATOR;
_penalty = (97 * 100) /10000;
_penalty = 9700 / 10000;
_penalty = 0;
```

Proof of Concept:

Listing 6: PoC Code - Penalty Fee Bypass

```
function test_withdrawFeeBypassPoc() public {
    vm.startPrank(user1);

    uint256 preBalanceUser1 = testToken.balanceOf(user1);
    uint256 preBalanceUser2 = testToken.balanceOf(user2);

    testToken.approve(address(vault), 102);
    vault.mint(102, user1);

    vm.stopPrank();

    vm.startPrank(user2);
    testToken.approve(address(vault), 100e18);
    vault.deposit(100e18, user2);
    vm.stopPrank();

    vm.startPrank(user3);
    testToken.approve(address(vault), 10e18);
    vault.deposit(10e18, user2);
    vm.stopPrank();


    vm.prank(user2);
    //vault.redeem(110, user2, user2); // @audit - withdraws 109 shares -> 110 - 1 (penalty fee)
    vault.redeem(97, user2, user2); // @audit - withdraws 97 shares
}
```



```
Running 1 test for test/AuraMain.t.sol:AuraMainTest
[PASS] test_ERC4626PoC() (gas: 304812)
Test result: ok. 1 passed; 0 failed; finished in 1.46ms
```

Code Location:

Listing 7: GenericVault.sol (Line 151)

```
133 function _withdraw(uint256 _shares) internal returns (uint256
  ↳ _withdrawable) {
134     require(totalSupply() > 0);
135     // Computes the amount withdrawable based on the number of
  ↳ shares sent
136     uint256 amount = (_shares * totalUnderlying()) / totalSupply()
  ↳ ;
137     // Burn the shares before retrieving tokens
138     _burn(msg.sender, _shares);
139     // If user is last to withdraw, harvest before exit
140     if (totalSupply() == 0) {
141         harvest();
142         IStrategy(strategy).withdraw(totalUnderlying());
143         _withdrawable = IERC20(underlying).balanceOf(address(this)
  ↳ );
144     }
145     // Otherwise compute share and unstake
146     else {
147         _withdrawable = amount;
148         // Substract a small withdrawal fee to prevent users "
  ↳ timing"
149         // the harvests. The fee stays staked and is therefore
150         // redistributed to all remaining participants.
151         uint256 _penalty = (_withdrawable * withdrawalPenalty) /
  ↳ FEE_DENOMINATOR;
152         _withdrawable = _withdrawable - _penalty;
153         IStrategy(strategy).withdraw(_withdrawable);
154     }
155     return _withdrawable;
156 }
```

Risk Level:

Likelihood - 4
Impact - 2

Remediation Plan:

SOLVED: The Aura Finance team solved the issue by changing their penalty fee formula.

Commit ID: diff-e4193a9b0001. . . 5bb4d6e0ee04R360

3.3 (HAL-03) THE LAST HARVEST OPERATION CAN BE SANDWICHED - LOW

Description:

If the totalSupply() function returns zero, the AuraBalVault:harvest() function can be called by anyone. Therefore, the harvest function can be called with zero as minAmtOut which is a slippage control parameter provided to the swap() function.

This scenario is very unlikely. This vulnerability affects the last harvester in the protocol.

In addition, the last user to harvest can also claim all remaining assets in the vault.

Code Location:

Listing 8: AuraBalVault.sol (Lines 42,59)

```
function harvest(uint256 _minAmountOut, bool _lock) public {
    require(
        !isHarvestPermissioned || authorizedHarvesters[msg.sender] || totalSupply() == 0,
        "permissioned harvest"
    );
    uint256 _harvested = AuraBalStrategy(strategy).harvest(msg.sender, _minAmountOut, _lock);
    emit Harvest(msg.sender, _harvested);
}

/// @notice Claim rewards and swaps them to auraBAL for restaking
/// @param _minAmountOut - min amount of BPT to receive for harvest
/// @dev locking for auraBAL by default
function harvest(uint256 _minAmountOut) public {
    harvest(_minAmountOut, true);
}

```

```
/// @notice Claim rewards and swaps them to auraBAL for restaking
/// @dev No slippage protection, swapping for auraBAL
function harvest() public override {
    harvest(0);
}
```

Listing 9: Strategy.sol (Line 161)

```
function harvest(uint256 _minAmountOut) public onlyVault returns (
↳ uint256 harvested) {
    // claim rewards
    auraBalStaking.getReward();

    // process extra rewards
    uint256 extraRewardCount = IGenericVault(vault).
↳ extraRewardsLength();
    for (uint256 i; i < extraRewardCount; ++i) {
        address rewards = IGenericVault(vault).extraRewards(i);
        address token = IVirtualRewards(rewards).rewardToken();
        uint256 balance = IERC20(token).balanceOf(address(this));
        if (balance > 0) {
            IERC20(token).safeTransfer(rewards, balance);
            IVirtualRewards(rewards).queueNewRewards(balance);
        }
    }

    // process rewards
    address[] memory _rewardTokens = rewardTokens;
    for (uint256 i; i < _rewardTokens.length; ++i) {
        address _tokenHandler = rewardHandlers[_rewardTokens[i]];
        if (_tokenHandler == address(0)) {
            continue;
        }
        uint256 _tokenBalance = IERC20(_rewardTokens[i]).balanceOf
↳ (address(this));
        if (_tokenBalance > 0) {
            IERC20(_rewardTokens[i]).safeTransfer(_tokenHandler,
↳ _tokenBalance);
            IRewardHandler(_tokenHandler).sell();
        }
    }

    uint256 _wethBalance = IERC20(WETH_TOKEN).balanceOf(address(
↳ this));
```

```
    uint256 _balBalance = IERC20(BAL_TOKEN).balanceOf(address(this));

    if (_wethBalance + _balBalance == 0) {
        return 0;
    }
    // Deposit to BLP
    _depositToBalEthPool(_balBalance, _wethBalance, 0);

    // Swap the LP tokens for aura BAL
    uint256 _bptBalance = IERC20(BAL_ETH_POOL_TOKEN).balanceOf(address(this));
    _swapBptToAuraBal(_bptBalance, _minAmountOut);

    uint256 _auraBalBalance = IERC20(AURABAL_TOKEN).balanceOf(address(this));
    if (_auraBalBalance > 0) {
        stake(_auraBalBalance);
        return _auraBalBalance;
    }

    return 0;
}
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to remove hardcoded slippage from contracts to prevent sandwich attacks occurring.

Remediation Plan:

RISK ACCEPTED: The harvest() function can be front run if at least one of the below statements is true:

1. the harvest is not permissioned, or
2. the caller is an authorized address, or
3. this is the last harvest.

The Aura Finance team accepted the risk and explained that

> We are not going to have anybody else permissioned to do harvest. So, isHarvestPermissioned is always going to be true. And, we will send 1 token to a dead address so totalSupply will never be 0. So, the harvest with no min out can never be called.

3.4 (HAL-04) DUPLICATE REWARD TOKENS CAN BE ADDED - LOW

Description:

There is no check on addRewardToken() function to prevent adding the same token more than one.

The addRewardToken() function should update the reward token parameters if it already exists. If the token is new, the new token should be pushed into the rewardTokens array directly.

Code Location:

Listing 10: Strategy.sol (Line 79)

```
function addRewardToken(address _token, address _handler) external
↳  onlyOwner {
    rewardTokens.push(_token);
    _updateRewardToken(_token, _handler);
}
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

It is suggested to add a require() check that controls if specified token already exists on the contract.

Remediation Plan:

RISK ACCEPTED: The Aura Finance team accepted the risk of the finding, and they will prefer using the current pattern on their protocol.

3.5 (HAL-05) LACK OF TWO-STEP OWNERSHIP PATTERN - LOW

Description:

To change the owner address, the current contract owner can call the Ownable.transferOwnership() function and set a new address and this new address assumes the role immediately.

If the new address is inactive or not willing to act in the role, there is no way to restore access to that role. Therefore, the owner role can be lost.

Code Location:

Listing 11: Ownable Contracts

```
Strategy.sol#L19
GenericVault.sol#L22
FeeForwarder.sol#L21
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

It is recommended to use the following Ownable2Step library instead of Ownable library:

Ownable2Step.sol

Remediation Plan:

RISK ACCEPTED: The Aura Finance team accepted the risk of the finding, and they will prefer using the current pattern on their protocol.

3.6 (HAL-06) MISTAKENLY SEND ETHERS LOCKED FOREVER IN VAULTS - INFORMATIONAL

Description:

The StrategyBase contract implements the receive() function to accept ether transfers. However, that contract does not use ETH for any operation. Therefore, if someone sends ETH to this contract, then ETH will be stuck in the contract forever.

Code Location:

Listing 12: StrategyBase.sol (Line 112)

```
112 receive() external payable {}
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Consider removing the receive() function to prevent accidental ETH transfers.

Remediation Plan:

SOLVED: The Aura Finance team solved this issue by removing the receive() function from StrategyBase contract.

Commit ID: diff-b0b5bba9e045. . . 00ccb6eba7dcL112

3.7 (HAL-07) FOR LOOP OPTIMIZATIONS - INFORMATIONAL

Description:

It has been observed all for loops in the protocol are not optimized. Suboptimal for loops can cost too much gas.

These for loops can be optimized with the suggestions above:

1. In Solidity (pragma 0.8.0 and later), adding the unchecked keyword for arithmetical operations can reduce gas usage on contracts where underflow/underflow is unrealistic. It is possible to save gas by using this keyword on multiple code locations.
2. In all for loops, the index variable is incremented using +=. It is known that, in loops, using ++i costs less gas per iteration than +=. This also affects incremented variables within the loop code block.
3. Do not initialize index variables with 0. Solidity already initializes these uint variables as zero.

Code Location:

Listing 13: Suboptimal For Loops

```
GenericVault.sol:#L99
GenericVault.sol:#L146
GenericVault.sol:#L209
GenericVault.sol:#L245
GenericVault.sol:#L250
Strategy.sol:#L126
Strategy.sol:#L138
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to apply the following pattern for Solidity pragma version 0.8.0 and later.

Listing 14: For Loop Optimization

```
for (uint256 i; i < arrayLength; ) {
    . . .
    unchecked {
     ++i
    }
```

Remediation Plan:

ACKNOWLEDGED: The Aura Finance team acknowledged this issue.

3.8 (HAL-08) REDUNDANT CODE - INFORMATIONAL

Description:

The GenericUnionVault._beforeTokenTransfer() function uses a redundant for loop. That function has two different for loops. Both for loops use the same expressions. It might be a better idea to use only one for loop for withdraw() and stake() functions.

Code Location:

Listing 15: GenericUnionVault.sol (Lines 245,250)

```
238 function _beforeTokenTransfer(
239     address from,
240     address to,
241     uint256 amount
242 ) internal override {
243     // Withdraw extra rewards for the "from" address to update their earned
244     // amount when updateReward is called
245     for (uint256 i = 0; i < extraRewards.length; i++) {
246         IBasicRewards(extraRewards[i]).withdraw(from, amount);
247     }
248
249     // Stake extra rewards for the "to" address
250     for (uint256 i = 0; i < extraRewards.length; i++) {
251         IBasicRewards(extraRewards[i]).stake(to, amount);
252     }
253 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider using only one for loop for withdraw() and stake() operations to decrease gas consumption.

Remediation Plan:

ACKNOWLEDGED: The Aura Finance team acknowledged this issue.

3.9 (HAL-09) INCORRECT COMMENTS - INFORMATIONAL

Description:

There are two typos in @dev notes.

Listing 16: AuraBalVault.sol - Typo 1 (Line 37)

```
/// @dev Harvest can be called even if permissioned when last
    staker is
/// withdrawing from the vault.
```

The comment says harvest can be called even **if permissioned** when the last staker is withdrawing from the vault. However, it should be corrected as **if not permissioned** since the protocol actually does not check permission for the last staker.

Listing 17: Strategy.sol - Typo 2 (Line 17)

```
*           - remove paltform fee
```

The second one should be corrected as remove platform fee.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is important to correct these typos for improving the readability of the code.

Remediation Plan:

ACKNOWLEDGED: The Aura Finance team acknowledged this issue.

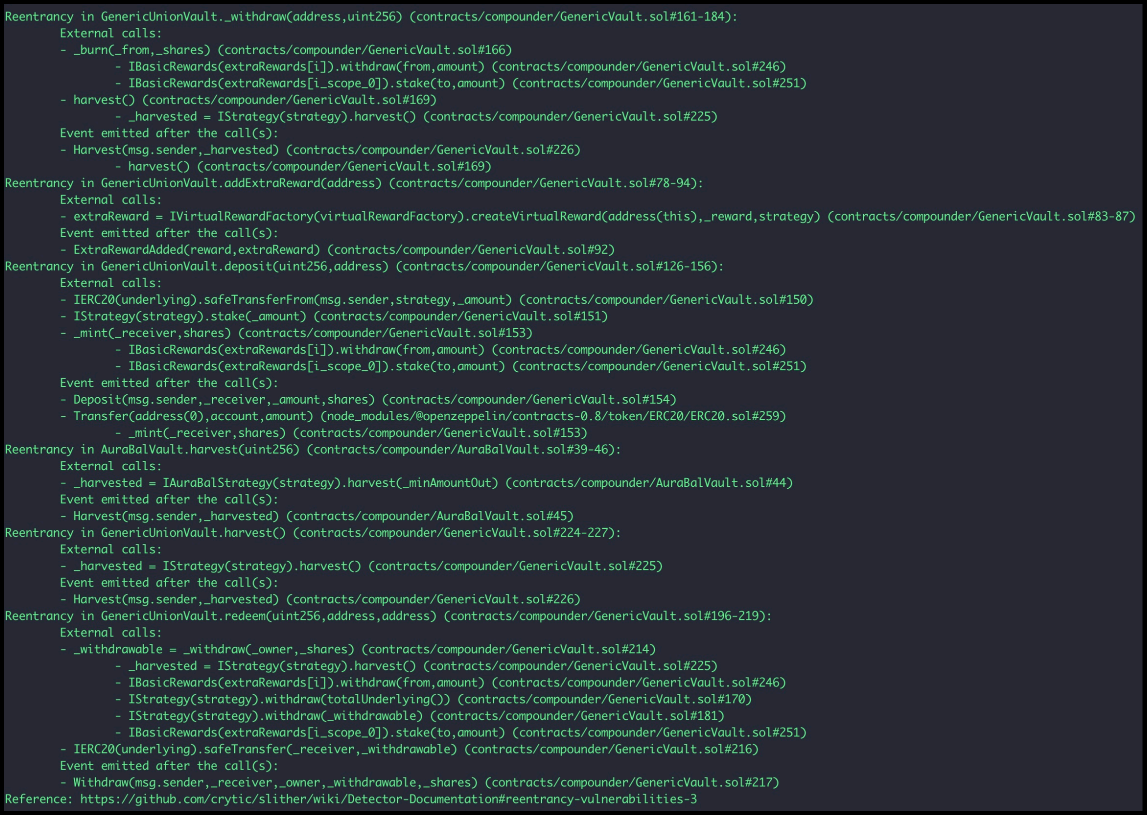
AUTOMATED TESTING

4.1 STATIC ANALYSIS SCAN

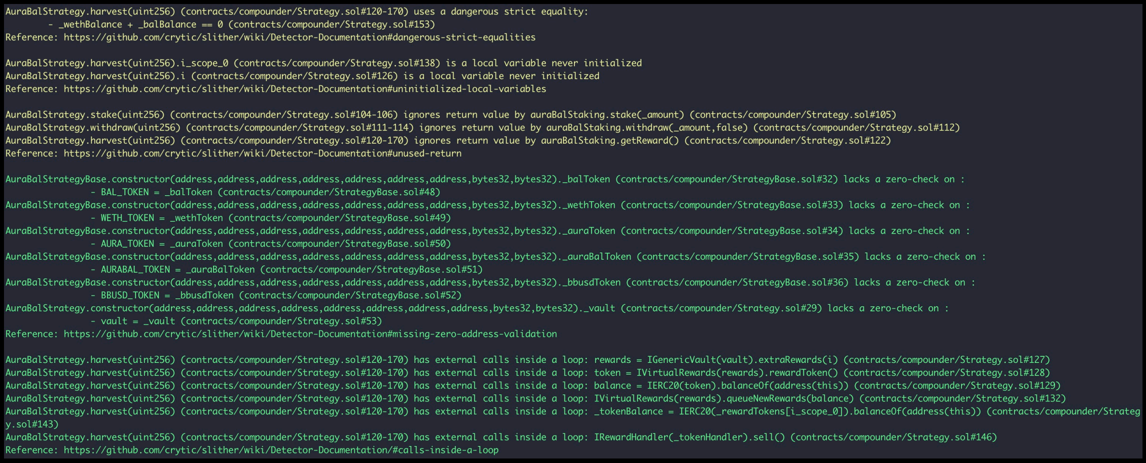
Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Slither, a Solidity static analysis framework, was used for static analysis. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

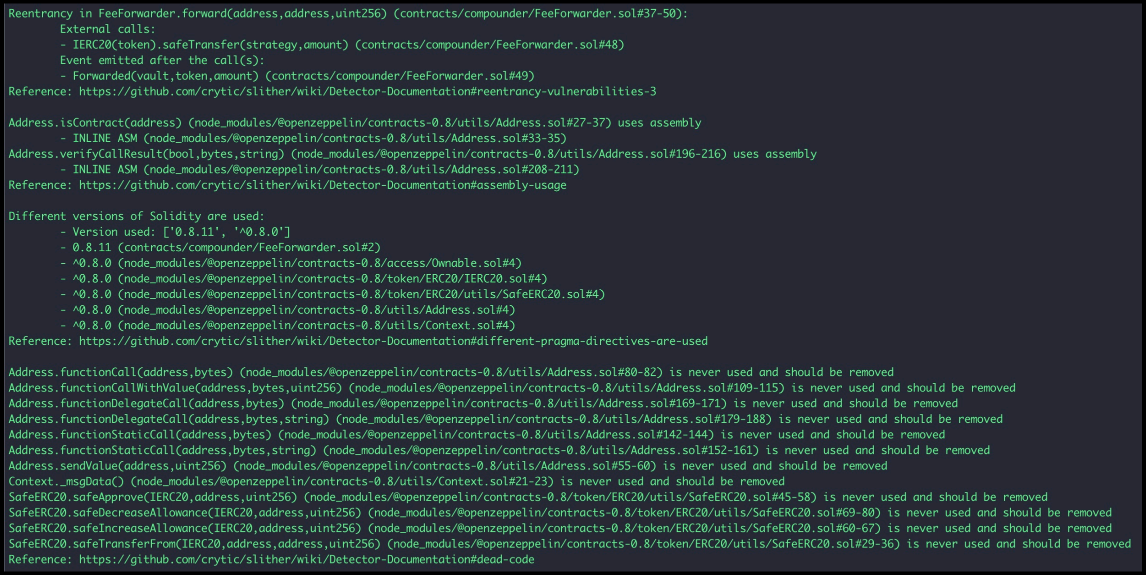
Results:



```
Reentrancy in GenericUnionVault._withdraw(address,uint256) (contracts/compounder/GenericVault.sol#161-184):
        External calls:
        - _burn(_from,_shares) (contracts/compounder/GenericVault.sol#166)
                - IBasicRewards(extraRewards[i]).withdraw(from,amount) (contracts/compounder/GenericVault.sol#246)
                - IBasicRewards(extraRewards[i_scope_0]).stake(to,amount) (contracts/compounder/GenericVault.sol#251)
        - harvest() (contracts/compounder/GenericVault.sol#169)
                - _harvested = IStrategy(strategy).harvest() (contracts/compounder/GenericVault.sol#225)
        Event emitted after the call(s):
        - Harvest(msg.sender,_harvested) (contracts/compounder/GenericVault.sol#226)
                - harvest() (contracts/compounder/GenericVault.sol#169)
Reentrancy in GenericUnionVault.addExtraReward(address) (contracts/compounder/GenericVault.sol#78-94):
        External calls:
        - extraReward = IVirtualRewardFactory(virtualRewardFactory).createVirtualReward(address(this),_reward,strategy) (contracts/compounder/GenericVault.sol#83-87)
        Event emitted after the call(s):
        - ExtraRewardAdded(reward,extraReward) (contracts/compounder/GenericVault.sol#92)
Reentrancy in GenericUnionVault.deposit(uint256,address) (contracts/compounder/GenericVault.sol#126-156):
        External calls:
        - IERC20(underlying).safeTransferFrom(msg.sender,strategy,_amount) (contracts/compounder/GenericVault.sol#150)
        - IStrategy(strategy).stake(_amount) (contracts/compounder/GenericVault.sol#151)
        - _mint(_receiver,shares) (contracts/compounder/GenericVault.sol#153)
                - IBasicRewards(extraRewards[i]).withdraw(from,amount) (contracts/compounder/GenericVault.sol#246)
                - IBasicRewards(extraRewards[i_scope_0]).stake(to,amount) (contracts/compounder/GenericVault.sol#251)
        Event emitted after the call(s):
        - Deposit(msg.sender,_receiver,_amount,shares) (contracts/compounder/GenericVault.sol#154)
        - Transfer(address(0),account,amount) (node_modules/@openzeppelin/contracts-0.8/token/ERC20/ERC20.sol#259)
                - _mint(_receiver,shares) (contracts/compounder/GenericVault.sol#153)
Reentrancy in AuraBalVault.harvest(uint256) (contracts/compounder/AuraBalVault.sol#39-46):
        External calls:
        - _harvested = IAuraBalStrategy(strategy).harvest(_minAmountOut) (contracts/compounder/AuraBalVault.sol#44)
        Event emitted after the call(s):
        - Harvest(msg.sender,_harvested) (contracts/compounder/AuraBalVault.sol#45)
Reentrancy in GenericUnionVault.harvest() (contracts/compounder/GenericVault.sol#224-227):
        External calls:
        - _harvested = IStrategy(strategy).harvest() (contracts/compounder/GenericVault.sol#225)
        Event emitted after the call(s):
        - Harvest(msg.sender,_harvested) (contracts/compounder/GenericVault.sol#226)
Reentrancy in GenericUnionVault.redeem(uint256,address,address) (contracts/compounder/GenericVault.sol#196-219):
        External calls:
        - _withdrawable = _withdraw(_owner,_shares) (contracts/compounder/GenericVault.sol#214)
                - _harvested = IStrategy(strategy).harvest() (contracts/compounder/GenericVault.sol#225)
                - IBasicRewards(extraRewards[i]).withdraw(from,amount) (contracts/compounder/GenericVault.sol#246)
                - IStrategy(strategy).withdraw(totalUnderlying()) (contracts/compounder/GenericVault.sol#170)
                - IStrategy(strategy).withdraw(_withdrawable) (contracts/compounder/GenericVault.sol#181)
                - IBasicRewards(extraRewards[i_scope_0]).stake(to,amount) (contracts/compounder/GenericVault.sol#251)
        - IERC20(underlying).safeTransfer(_receiver,_withdrawable) (contracts/compounder/GenericVault.sol#216)
        Event emitted after the call(s):
        - Withdraw(msg.sender,_receiver,_owner,_withdrawable,_shares) (contracts/compounder/GenericVault.sol#217)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```



```
AuraBalStrategy.harvest(uint256) (contracts/compounder/Strategy.sol#120-170) uses a dangerous strict equality:
	- _wethBalance + _balBalance == 0 (contracts/compounder/Strategy.sol#153)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

AuraBalStrategy.harvest(uint256).i_scope_0 (contracts/compounder/Strategy.sol#138) is a local variable never initialized
AuraBalStrategy.harvest(uint256).i (contracts/compounder/Strategy.sol#126) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

AuraBalStrategy.stake(uint256) (contracts/compounder/Strategy.sol#104-106) ignores return value by auraBalStaking.stake(_amount) (contracts/compounder/Strategy.sol#105)
AuraBalStrategy.withdraw(uint256) (contracts/compounder/Strategy.sol#111-114) ignores return value by auraBalStaking.withdraw(_amount,false) (contracts/compounder/Strategy.sol#112)
AuraBalStrategy.harvest(uint256) (contracts/compounder/Strategy.sol#120-170) ignores return value by auraBalStaking.getReward() (contracts/compounder/Strategy.sol#122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

AuraBalStrategyBase.constructor(address,address,address,address,address,address,address,bytes32,bytes32)._balToken (contracts/compounder/StrategyBase.sol#32) lacks a zero-check on :
		- BAL_TOKEN = _balToken (contracts/compounder/StrategyBase.sol#48)
AuraBalStrategyBase.constructor(address,address,address,address,address,address,address,bytes32,bytes32)._wethToken (contracts/compounder/StrategyBase.sol#33) lacks a zero-check on :
		- WETH_TOKEN = _wethToken (contracts/compounder/StrategyBase.sol#49)
AuraBalStrategyBase.constructor(address,address,address,address,address,address,address,bytes32,bytes32)._auraToken (contracts/compounder/StrategyBase.sol#34) lacks a zero-check on :
		- AURA_TOKEN = _auraToken (contracts/compounder/StrategyBase.sol#50)
AuraBalStrategyBase.constructor(address,address,address,address,address,address,address,bytes32,bytes32)._auraBalToken (contracts/compounder/StrategyBase.sol#35) lacks a zero-check on :
		- AURABAL_TOKEN = _auraBalToken (contracts/compounder/StrategyBase.sol#51)
AuraBalStrategyBase.constructor(address,address,address,address,address,address,address,bytes32,bytes32)._bbusdToken (contracts/compounder/StrategyBase.sol#36) lacks a zero-check on :
		- BBUSD_TOKEN = _bbusdToken (contracts/compounder/StrategyBase.sol#52)
AuraBalStrategy.constructor(address,address,address,address,address,address,address,address,bytes32,bytes32)._vault (contracts/compounder/Strategy.sol#29) lacks a zero-check on :
		- vault = _vault (contracts/compounder/Strategy.sol#53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

AuraBalStrategy.harvest(uint256) (contracts/compounder/Strategy.sol#120-170) has external calls inside a loop: rewards = IGenericVault(vault).extraRewards(i) (contracts/compounder/Strategy.sol#127)
AuraBalStrategy.harvest(uint256) (contracts/compounder/Strategy.sol#120-170) has external calls inside a loop: token = IVirtualRewards(rewards).rewardToken() (contracts/compounder/Strategy.sol#128)
AuraBalStrategy.harvest(uint256) (contracts/compounder/Strategy.sol#120-170) has external calls inside a loop: balance = IERC20(token).balanceOf(address(this)) (contracts/compounder/Strategy.sol#129)
AuraBalStrategy.harvest(uint256) (contracts/compounder/Strategy.sol#120-170) has external calls inside a loop: IVirtualRewards(rewards).queueNewRewards(balance) (contracts/compounder/Strategy.sol#132)
AuraBalStrategy.harvest(uint256) (contracts/compounder/Strategy.sol#120-170) has external calls inside a loop: _tokenBalance = IERC20(_rewardTokens[i_scope_0]).balanceOf(address(this)) (contracts/compounder/Strateg
y.sol#143)
AuraBalStrategy.harvest(uint256) (contracts/compounder/Strategy.sol#120-170) has external calls inside a loop: IRewardHandler(_tokenHandler).sell() (contracts/compounder/Strategy.sol#146)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
```



```
Reentrancy in FeeForwarder.forward(address,address,uint256) (contracts/compounder/FeeForwarder.sol#37-50):
	External calls:
	- IERC20(token).safeTransfer(strategy,amount) (contracts/compounder/FeeForwarder.sol#48)
	Event emitted after the call(s):
	- Forwarded(vault,token,amount) (contracts/compounder/FeeForwarder.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Address.isContract(address) (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#27-37) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#33-35)
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#196-216) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#208-211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
	- Version used: ['0.8.11', '^0.8.0']
	- 0.8.11 (contracts/compounder/FeeForwarder.sol#2)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-0.8/access/Ownable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-0.8/token/ERC20/IERC20.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-0.8/token/ERC20/utils/SafeERC20.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts-0.8/utils/Context.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#80-82) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#109-115) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#169-171) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#179-188) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#142-144) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#152-161) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-0.8/utils/Address.sol#55-60) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts-0.8/utils/Context.sol#21-23) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (node_modules/@openzeppelin/contracts-0.8/token/ERC20/utils/SafeERC20.sol#45-58) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts-0.8/token/ERC20/utils/SafeERC20.sol#69-80) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts-0.8/token/ERC20/utils/SafeERC20.sol#60-67) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (node_modules/@openzeppelin/contracts-0.8/token/ERC20/utils/SafeERC20.sol#29-36) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

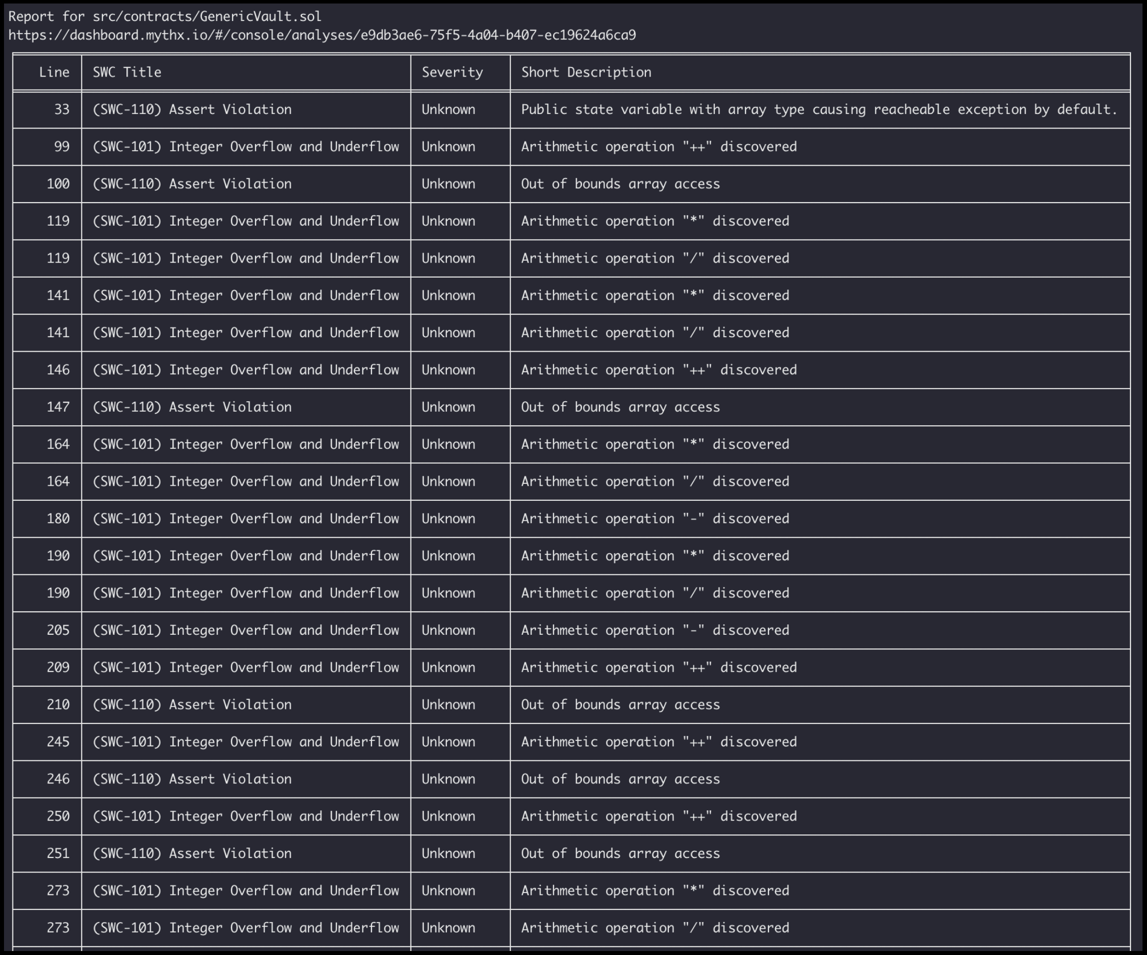
As a result of the tests carried out with the Slither tool, some results were obtained and these results were reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives and these results were not included in the report. The actual vulnerabilities found by Slither are already included in the report findings.

4.2 AUTOMATED SECURITY SCAN

Description:

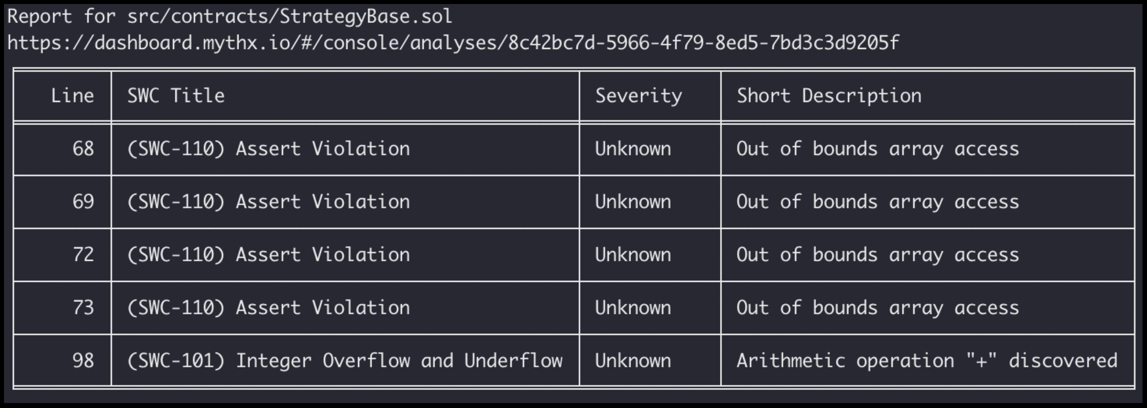
Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:



```
Report for src/contracts/GenericVault.sol
https://dashboard.mythx.io/#/console/analyses/e9db3ae6-75f5-4a04-b407-ec19624a6ca9

| Line | SWC Title                                  | Severity | Short Description                                                        |
|   33 | (SWC-110) Assert Violation                 | Unknown  | Public state variable with array type causing reacheable exception by default. |
|   99 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "++" discovered                                     |
|  100 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                               |
|  119 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "*" discovered                                      |
|  119 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "/" discovered                                      |
|  141 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "*" discovered                                      |
|  141 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "/" discovered                                      |
|  146 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "++" discovered                                     |
|  147 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                               |
|  164 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "*" discovered                                      |
|  164 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "/" discovered                                      |
|  180 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-" discovered                                      |
|  190 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "*" discovered                                      |
|  190 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "/" discovered                                      |
|  205 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "-" discovered                                      |
|  209 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "++" discovered                                     |
|  210 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                               |
|  245 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "++" discovered                                     |
|  246 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                               |
|  250 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "++" discovered                                     |
|  251 | (SWC-110) Assert Violation                 | Unknown  | Out of bounds array access                                               |
|  273 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "*" discovered                                      |
|  273 | (SWC-101) Integer Overflow and Underflow   | Unknown  | Arithmetic operation "/" discovered                                      |
```

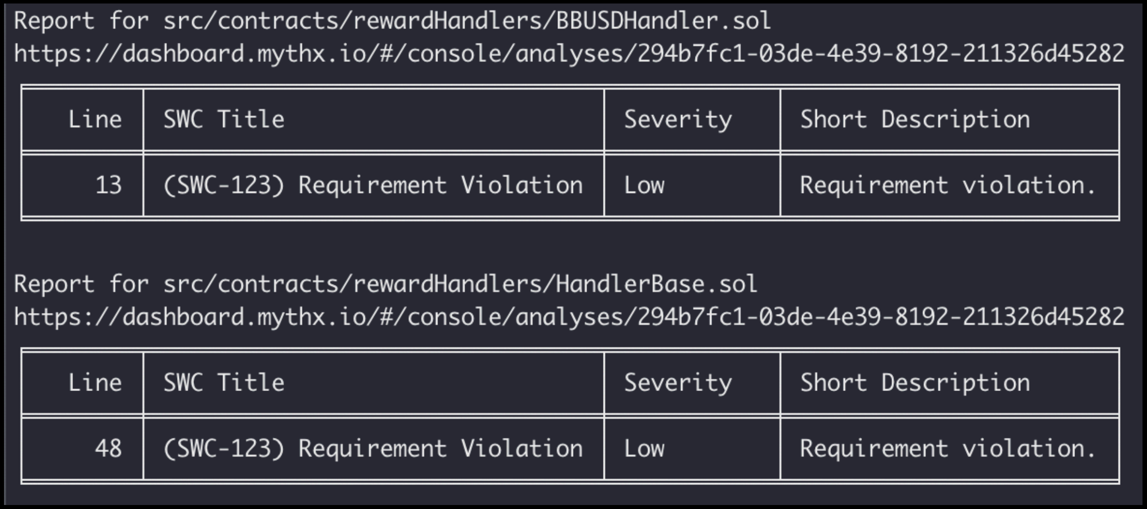
Figure 1: MythX Result - 1



```
Report for src/contracts/StrategyBase.sol
https://dashboard.mythx.io/#/console/analyses/8c42bc7d-5966-4f79-8ed5-7bd3c3d9205f
```

Line	SWC Title	Severity	Short Description
68	(SWC-110) Assert Violation	Unknown	Out of bounds array access
69	(SWC-110) Assert Violation	Unknown	Out of bounds array access
72	(SWC-110) Assert Violation	Unknown	Out of bounds array access
73	(SWC-110) Assert Violation	Unknown	Out of bounds array access
98	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered

Figure 2: MythX Result - 2



```
Report for src/contracts/rewardHandlers/BBUSDHandler.sol
https://dashboard.mythx.io/#/console/analyses/294b7fc1-03de-4e39-8192-211326d45282
```

Line	SWC Title	Severity	Short Description
13	(SWC-123) Requirement Violation	Low	Requirement violation.

```
Report for src/contracts/rewardHandlers/HandlerBase.sol
https://dashboard.mythx.io/#/console/analyses/294b7fc1-03de-4e39-8192-211326d45282
```

Line	SWC Title	Severity	Short Description
48	(SWC-123) Requirement Violation	Low	Requirement violation.

Figure 3: MythX Result - 3

The findings obtained as a result of the MythX scan were examined, and the findings were not included in the report because they were false positive.

THANK YOU FOR CHOOSING

HALBORN