SECURITY AUDIT OF

PIT FINANCE SMART CONTRACT

Public Report

Sep 12, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Sei	Sei is an open-source, layer-1 blockchain offering infrastructure for trading apps of different types, including non-fungible token marketplaces, gaming DEXs, and DeFi DEXs. The chain conducts market-based parallelization and has a native order-matching engine that allows exchange teams to use Sei's twin-turbo consensus.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Sep 12, 2024. We would like to thank the Pit Finance for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Pit Finance Smart Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified no vulnerable issue in the smart contracts code which was almost similar to the original `Yearn` protocol.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Pit Finance Smart Contract

Pit Finance is a decentralized yield optimizer built exclusively for the Sei Network, designed to maximize the potential of crypto assets. Leveraging advanced strategies and technology, Pit Finance enables users and protocols to deposit digital assets and earn optimized yields while prioritizing security and efficiency within the Sei ecosystem.

At the core of Pit Finance are our Vaults—automated tools that compound rewards from decentralized exchanges and other DeFi platforms back into the user's initially deposited assets. Importantly, your funds are never locked in any contract on Pit Finance, allowing you to withdraw at any time.

1.2. Audit Scope

The Pit Finance Smart Contract is forked from `Yearn` protocol. This audit focused on identifying security of different codebases from the original which was deployed on the Sei Network.

The latest version was made available in the course of the review:

Contract	Address	Is Proxy	Implementation Address
PitVaultV3	0x5512D5E89a29447462Ab6BBBcbD3fe6F0A0D9FDd	False	
PitVaultV3Factory	0x6944BfDAcD00957715eCBCeAac3F49e07cB6f35F	False	
TokenizedStrategy	0x7c96Bf5203fA1584A81f160B9445EAB02bfa3E7B	False	
Registry	0xE560fC83EA78028a95CCAb1ECD45039fFB62301C	False	
ReleaseRegistry	0x43e60c59567996C5a1b27ca8719ADF6CA2bC407B	False	
Accountant	0x79756b033390232a5A55B5D6571a7Bc1b11A2b93	False	

Contract	Address	Is Proxy	Implementation Address
AccountantFactory	0x07428D1CdC20Ce70Ac5D06E44CCB9920F99ab447	False	
Keeper	0x9CDdFFc79Ba597CCBBd6da6a81Cc1b1ef29FB4de	False	
AprOracle	0xed6A3CD7591Df25bC92E42381d9c53E041B5ACAa	False	
Aave3StrategyAprOracle	0xdF7F9020800c2B444ad524982Fc6cA5A30915811	False	
eip1167	0x63E37028c1740303e8456962E6ddf98359FE0BDc	True	0x5512D5E89a29447462Ab6BBBcbD3fe6F0A0D9FDd
PitAaveV3SimpleLender	0x0044f5f3F121beB7ef11c6cfBc45a56e80adFd6B	False	
eip1167	0x1e3BcEb9AD3dc3f80820d29039C1A46e28d3A573	True	0x5512D5E89a29447462Ab6BBBcbD3fe6F0A0D9FDd
PitAaveV3SimpleLender	0x5E2BAF273586F747e406eB0fD0CAdAECe0f479Da	False	
eip1167	0x575E2291C6FDe8eEf7284CBd7Ff17615de75c47B	True	0x5512D5E89a29447462Ab6BBBcbD3fe6F0A0D9FDd
PitAaveV3SimpleLender	0x1eB5B573852897831414Fbd607c214EdB9b0B7Dd	False	
eip1167	0x49561C4a905f7acCdaCAec5e3C17113d5f1C5a3b	True	0x5512D5E89a29447462Ab6BBBcbD3fe6F0A0D9FDd
PitAaveV3SimpleLender	0xA4A3B54665A5ef21ac5B294b8945Dd130aC6E5fc	False	
eip1167	0xE8d7f53173BB5D0C898e4D576E4E297cB2859472	True	0x5512D5E89a29447462Ab6BBBcbD3fe6F0A0D9FDd

Contract	Address	Is Proxy	Implementation Address
PitAaveV3SimpleLender	0xCd6d59a116F0d4E370A62b8D6EA146dD02E2114A	False	
eip1167	0x0AEfC6F2E9a866ddB4813Cb1E897a2E9e26a1E53	True	0x5512D5E89a29447462Ab6BBBcbD3fe6F0A0D9FDd
PitAaveV3SimpleLender	0xE77790035f956E9C6E895e57cded63A7AC69F2c9	False	

Table 1. The scope of the audit

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 2. Severity levels

1.4. Disclaimer

Pit Finance acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Pit Finance understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Pit Finance agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Pit Finance will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Pit Finance, the final report will be considered fully accepted by the Pit Finance without the signature.

2. AUDIT RESULT

2.1. Overview

The Pit Finance Smart Contract was coded in `Solidity` (0.8.18) and `Vyper` (0.3.7) languages.

The `Vault`, `Keeper`, `Accountant`, `Registry`, and `AprOracle` modules remain the same as in the original `Yearn` protocol. However, the strategy for compounding (via the `PitAaveV3SimpleLender` contract) has been modified. The `PitAaveV3SimpleLender` contract removed certain caution-related functions and adjusted the reward claiming mechanism. Now, when users claim rewards, in addition to the original rewards, they receive additional rewards, all of which are converted to the `asset` token via `Uniswap` V2/V3 router. The `diff` is shown below:

```
@@ -1,14 +1,16 @@
-contract AaveV3Lender is BaseStrategy, UniswapV3Swapper, AuctionSwapper {
+contract PitAaveV3SimpleLender is BaseStrategy, UniswapUnifiedSwapper {
…
-    IPool public constant lendingPool =
-        IPool(0x87870Bca3F3fD6335C3F4ce8392D69350B4fA4E2);
+    IPool private immutable lendingPool;

-    IStakedAave internal constant stkAave =
-        IStakedAave(0x4da27a545c0c5B758a6BA100e3a049001de870f5);
-    address internal constant AAVE =
-        address(0x7Fc66500c84A76Ad7e9c93437bFc5Ac33E2DDaE9);
-
 …

-    // If rewards should be sold through Auctions.
-    bool public useAuction = true;
-
…

     constructor(
         address _asset,
-        string memory _name
-    ) BaseStrategy(_asset, _name) {
+        string memory _name,
+        address _tokenized_strategy,
+        address _lendingPool,
+        address _weth,
+        address _uniRouter,
+        address _uniV3Router
+    ) BaseStrategy(_asset, _name, _tokenized_strategy) {
+        // Set the lending pool.
+        lendingPool = IPool(_lendingPool);
+
…
-        // Get aToken decimals for supply caps.
-           decimals = ERC20(address(aToken)).decimals();
-
            // Set the rewards controller
            rewardsController = aToken.getIncentivesController();

+           // Get aToken decimals for supply caps.
+           decimals = ERC20(address(aToken)).decimals();
+
…
-           base = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
-           router = 0xE592427A0AEce92De3Edee1F18E0157C05861564;
+           base = _weth;
+           router = _uniRouter;
+           v3Router = _univ3Router;
        }

        /**
+        * @dev Sets the additional rewards contract.
+        * Can only be called by the contract owner.
+        * @param _rewardsEscrow The address of the additional rewards contract.
+        */
+       function setRewardsEscrow(address _rewardsEscrow) external onlyManagement {
+           additionalRewards = IRewardsEscrow(_rewardsEscrow);
+       }
+
…
+       /**
+        * @notice Set the router to use for swaps.
+        * @dev External function available to management to set the
+        * Uni router to use for swaps.
+        *
+        * @param _v2router The V2 router to use.
+        * @param _v3router The V3 router to use.
+        */
+       function setRouter(
+           address _v2router,
+           address _v3router
+       ) external onlyManagement {
+           _setRouter(_v2router, _v3router);
+       }
+
+       /**
+        * @notice Configure a token to use V3 or V2 router.
+        * @dev External function available to management to set the
+        * Uni router version to use for swaps for a specific token.
+        *
+        * @param _token The token to configure.
+        * @param _useV3 Whether to use V3 or not.
+        */
+       function setUseV3Router(
+           address _token,
+        bool _useV3
+    ) external onlyManagement {
+        _setUseV3Router(_token, _useV3);
+    }
…
-    /**
-     * @notice Used to claim any pending rewards and sell them to asset.
-     */
     function _claimAndSellRewards() internal {
-        // Claim any pending stkAave.
-        _redeemAave();
-
-        //claim all rewards
+        // Claim all rewards from lending pool
         address[] memory assets = new address[](1);
         assets[0] = address(aToken);
-        (address[] memory rewardsList, ) = rewardsController
+        (address[] memory aaveRewardsList, ) = rewardsController
             .claimAllRewardsToSelf(assets);
+        _sellRewardTokens(aaveRewardsList);

-        // Start cooldown on any new stkAave.
-        _harvestStkAave();
+        // Claim additional rewards if the additionalRewards contract is set
+        if (address(additionalRewards) != address(0)) {
+            address[] memory additionalRewardsList = additionalRewards
+                .claimAllAdditionalRewards();
+            _sellRewardTokens(additionalRewardsList);
+        }

-        // If using the Auction contract we are done.
-        if (useAuction) return;
+        // Redeposit rewards
+        uint256 currentBalance = balanceOfAsset();
+        if (currentBalance > 0) _depositToPool(currentBalance);
+    }

-        // Else swap as much as possible back to asset through uni.
+    function _sellRewardTokens(address[] memory rewardsList) private {
+        // Sell any rewards that are not `asset`
         address token;
         for (uint256 i = 0; i < rewardsList.length; ++i) {
             token = rewardsList[i];

-            if (token == address(asset)) {
+            if (token == address(asset)) {
                 continue;
-            } else if (token == address(stkAave)) {
-                // We swap Aave => asset
-                token = AAVE;
```

```
                }

                uint256 balance = ERC20(token).balanceOf(address(this));
-
                if (balance > minAmountToSellMapping[token]) {
                    _swapFrom(token, address(asset), balance, 0);
                }
            }
        }

-       function _redeemAave() internal {
-           if (!checkCooldown()) {
-               return;
-           }

-           uint256 stkAaveBalance = ERC20(address(stkAave)).balanceOf(
-               address(this)
-           );
-
-           if (stkAaveBalance > 0) {
-               stkAave.redeem(address(this), stkAaveBalance);
-           }
-       }
-
-       function checkCooldown() public view returns (bool) {
-           uint256 cooldownStartTimestamp = IStakedAave(stkAave).stakersCooldowns(
-               address(this)
-           );
-
-           if (cooldownStartTimestamp == 0) return false;
-
-           uint256 COOLDOWN_SECONDS = IStakedAave(stkAave).COOLDOWN_SECONDS();
-           uint256 UNSTAKE_WINDOW = IStakedAave(stkAave).UNSTAKE_WINDOW();
-           if (block.timestamp >= cooldownStartTimestamp + COOLDOWN_SECONDS) {
-               return
-                   block.timestamp - (cooldownStartTimestamp + COOLDOWN_SECONDS) <=
-                   UNSTAKE_WINDOW;
-           } else {
-               return false;
-           }
-       }
-
-       function _harvestStkAave() internal {
-           // request start of cooldown period
-           if (ERC20(address(stkAave)).balanceOf(address(this)) > 0) {
-               stkAave.cooldown();
-           }
-       }
-
-       function manualRedeemAave() external onlyKeepers {
-           _redeemAave();
-    }
-
…
-    ///////////// DUTCH AUCTION FUNCTIONS \\\\\\\\\\\\\\\\\\
-
-    function setAuction(address _auction) external onlyEmergencyAuthorized {
-        if (_auction != address(0)) {
-            require(Auction(_auction).want() == address(asset), "wrong want");
-        }
-        auction = _auction;
-    }
-
-    function _auctionKicked(
-        address _token
-    ) internal virtual override returns (uint256 _kicked) {
-        require(_token != address(asset), "asset");
-        _kicked = super._auctionKicked(_token);
-        require(_kicked >= minAmountToSellMapping[_token], "too little");
-    }
-
     /**
-     * @notice Set if tokens should be sold through the dutch auction contract.
-     */
-    function setUseAuction(bool _useAuction) external onlyManagement {
-        useAuction = _useAuction;
-    }
-
-    /**
      * @dev Optional function for a strategist to override that will
      * allow management to manually withdraw deployed funds from the
      * yield source if a strategy is shutdown.
```

2.2. Findings

During the audit process, the audit team found no vulnerability in the given version of Pit Finance Smart Contract.

2.3. Additional Notes and Recommendations

The audit team has identified some issues that could be improved in the code. The following recommendations should be considered to ensure the security and efficiency of the code.

2.3.1. [INFORMATIVE] - Missing `_minAmountOut` when swapping tokens

Positions:

- `PitAaveV3SimpleLender`#_sellRewardTokens()

Description:

When selling reward tokens, the code calls swapping with the `_minAmountOut` parameter is zero. It is recommended to allow the user to pass the minimum amount of tokens they expect to receive to prevent the front-running attack.

```
function _sellRewardTokens(address[] memory rewardsList) private {
        // Sell any rewards that are not `asset`
        address token;
        for (uint256 i = 0; i < rewardsList.length; ++i) {
            //...
            if (balance > minAmountToSellMapping[token]) {
                _swapFrom(token, address(asset), balance, 0);
            }
        }
    }

    function _swapFrom(
        address _from,
        address _to,
        uint256 _amountIn,
        uint256 _minAmountOut
    ) internal returns (uint256 _amountOut) {
        if (useV3Router[_from]) {
            _amountOut = _swapFromV3(_from, _to, _amountIn, _minAmountOut);
        } else {
            _amountOut =_swapFromV2(_from, _to, _amountIn, _minAmountOut);
        }
    }
```

2.3.2. [INFORMATIVE] - Unnecessary initialization of `base`, `router`, and `v3router` in `PitAaveV3SimpleLender` contract

Positions:

- `PitAaveV3SimpleLender`#constructor()

Description:

In the `PitAaveV3SimpleLender` contract, the `base`, `router`, and `v3router` variables are hardcoded. With the save gas cost, it is recommended to remove the initialization of these variables in the constructor.

```
address public base = 0xE30feDd158A2e3b13e9badaeABaFc5516e95e8C7;

    // Uni Swap router to use. Defaults to DragonSwap router on mainnet.
    address public router = 0x11DA6463D6Cb5a03411Dbf5ab6f6bc3997Ac7428;

    // Uni V3 Swap router to use. Defaults to DragonSwap router on mainnet.
    address public v3Router = 0xE592427A0AEce92De3Edee1F18E0157C05861564;

    constructor(
        //...
        address _weth,
        address _uniRouter,
        address _uniV3Router
    ) {
        //...
        base = _weth;
        router = _uniRouter;
        v3Router = _uniV3Router;
    }
```

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Sep 12, 2024*	Public Report	Verichains Lab

Table 3. Report versions history