// HALBORN

Oh!Finance

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **June 15th, 2021 - June 30th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/23/2021	Gabi Urrutia
0.2	Document Edits	06/24/2021	Oussama Amri
1.0	Document Edits	06/30/2021	Gabi Urrutia
1.1	Remediation Plan	08/13/2021	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
Oussama Amri	Halborn	Oussama.Amri@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Oh!Finance is a Decentralized Finance (DeFi) offering optimized yield-generating products, focusing on reducing risk and increasing volume exposure.

Oh!Finance engaged Halborn to conduct a security assessment on their smart contracts beginning on June 15th, 2021 and ending June 30th, 2021. The security assessment was scoped to smart contracts implementing the core protocol and the staking mechanism, and an audit of the security risk and implications regarding the changes introduced by the development team at Oh!Finance prior to its production release shortly following the assessments deadline.

1.2 AUDIT SUMMARY

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

The purpose of this audit to achieve the following:

- Ensure that smart contract functions are intended.
- Identify potential security issues with the smart contracts.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to 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 do not 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 (Hardhat and manual deployments on Ganache)
- Manual testing with custom Javascript.
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Testnet deployment (Remix IDE)

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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was 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.

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

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the smart contracts:

Oh!Finance:
- /contracts/bank/OhBank.sol
- /contracts/strategies/aave/OhAaveV2Strategy.sol
- /contracts/strategies/compound/OhCompoundStrategy.sol
- /contracts/strategies/curve/OhCurve3PoolStrategy.sol

Commit ID: aaa5e9eff8bf6f459f34d8c9e251af2254e078a4
Fixed Commit ID: 240a1a261b7f3a9e11031f7a078557073bc7b07d

OUT-OF-SCOPE:
Other smart contracts in the repository, external libraries and economics attacks.

However, if any economic issue is found, it will be marked as an INFORMATIONAL. This report identified several items that are economic in nature, (such as the way Liquidity can be accessed by owners) but may not be considered vulnerabilities in the context for this scope.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	5

LIKELIHOOD

IMPACT

		(HAL-01)		
(HAL-02) (HAL-03)				
(HAL-08)				
(HAL-04) (HAL-05) (HAL-06) (HAL-07)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - USE OF TX.ORIGIN	Medium	Low
HAL02 - UNCHECKED TRANSFER	Low	SOLVED - 08/04/2021
HAL03 - USE OF BLOCK.TIMESTAMP	Low	NOT APPLICABLE
HAL04 - MISSING EVENTS EMITTING	Informational	SOLVED - 08/06/2021
HAL05 - MISSING RE-ENTRANCY PROTECTION	Informational	RISK ACCEPTED
HAL06 - IMPRECISION OF A CONSTANT	Informational	ACKNOWLEDGED
HAL07 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	ACKNOWLEDGED
HAL08 - LACK OF LIQUIDITY LOSS PROTECTION	Informational	RISK ACCEPTED

FINDINGS & TECH DETAILS

3.1 (HAL-01) USE OF TX.ORIGIN - MEDIUM

Description:

OhBank.sol contract use tx.origin so that defense modifier can be called by anybody. It is recommended that you use msg.sender instead of tx.origin because if a transaction is made to a malicious wallet, when you check it you will have the origin address and you will not be able to know the address of the malicious wallet. Nevertheless, the use of tx.origin is semi-legitimized for recording who calls the contract most. Furthermore, tx.origin could be used to prevent an address from interacting with your contract because the owner of the address cannot use the contract as an intermediary to circumvent your blocking. Finally, it is important to remark that the use of tx.origin will be deprecated.

Code Location:

OhBank.sol Line #37

Listing 1: OhBank.sol (Lines 38)

```
37     modifier defense {
38         require(msg.sender == tx.origin || IManager(manager()).
               whitelisted(msg.sender), "Bank: Only EOA or whitelisted
               ");
39         _;
40     }
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended not to use tx.origin because a malicious wallet could receive funds and cannot be tracked. However, its use is semi-legitimate in some cases with caution.

Remediation Plan:

This modifier's purpose is to prevent smart contracts from interacting with
the Bank contract (i.e. malicious attacks). The only time that tx.origin == msg.sender is when the top-level caller is a user, not a contract. The OpenZeppelin implementation of the Address
library is not sufficient for identifying whether an address is NOT a contract, therefore this is the only form of logic currently available in Solidity that accurately blocks contract interaction. Contracts are upgradeable and logic can be updated if EIP-3074 is passed and implemented.

The issue is reclassified to **LOW**.

3.2 (HAL-02) UNCHECKED TRANSFER - LOW

Description:

The contracts OhBank.sol has _deposit method and in this method, transferFrom() is being called without any implementing checks on the return value. Several tokens do not revert in case of failure and return false which may allow an attacker to deposit for free.

Code Location:

Listing 2: OhBank.sol (Lines 225)

```
212  // deposit underlying to receive shares
213      function _deposit(
214          uint256 amount,
215          address sender,
216          address recipient
217      ) internal {
218          require(totalStrategies() > 0, "Bank: No Strategies");
219          require(amount > 0, "Bank: Invalid Deposit");
220
221          uint256 totalSupply = totalSupply();
222          uint256 mintAmount = totalSupply == 0 ? amount : amount.
                 mul(totalSupply).div(virtualBalance());
223
224          _mint(recipient, mintAmount);
225          IERC20(underlying()).transferFrom(sender, address(this),
                 amount);
226
227          emit Deposit(recipient, amount);
228      }
229
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Although `using SafeERC20 for IERC20;` is used we recommend using safe-TransferFrom() instead of the transferFrom() function.

Remediation Plan:

SOLVED: Updated OhBank.sol to use the OpenZeppelin `SafeERC20.safeTransferFrom` method. This method will cause any ERC20 to revert in case of failure. Fixed in coomit ID: d7ef893c0377d347b9730e724d70b5c7e7b6f4a1

3.3 (HAL-03) USE OF BLOCK.TIMESTAMP - LOW

Description:

During a manual static review, the tester noticed the use of block.timestamp in OhAaveV2Strategy.sol contract. The contract developers should be aware that this does not mean current time. Miners can influence the value of block.timestamp to perform Maximal Extractable Value (MEV) attacks. The use of now creates a risk that time manipulation can be performed to manipulate price oracles. Miners can modify the timestamp by up to 900 seconds.

Code Location:

Listing 3: OhAaveV2Strategy.sol (Lines 92,99)

```
87 /// @dev Compound stkAAVE rewards on a alternating cooldown
       schedule
88     function _compound() internal {
89         uint256 currentCooldown = rewardCooldown();
90
91         // if the current cooldown has passed
92         if (block.timestamp > currentCooldown) {
93             // save state variables
94             uint256 balance = stakedBalance();
95             address staked = stakedToken();
96             uint256 expiration = currentCooldown.add(
                   unstakingWindow(staked));
97
98             // if we have stkAAVE and the unstaking window hasn't
                   passed
99             if (balance > 0 && block.timestamp < expiration) {
100                 // redeem all available AAVE
101                 redeem(staked, balance);
102
103                 // validate we received AAVE
104                 uint256 amount = rewardBalance();
105                 if (amount > 0) {
106                     // liquidate for underlying
```

```
107                     liquidate(reward(), underlying(), amount);
108                 }
109             }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Use block.number instead of block.timestamp to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years, days and months rather than seconds. If possible, it is recommended to use Oracles.

Remediation Plan:

NOT APPLICABLE: The AaveV2Strategy.sol file only deals with time periods greater than 48 hours. There is a 480 hour waiting period that must first pass to convert stkAAVE into AAVE. Then there is a maximum 48 hour window in which the conversion can take place. Timeframes of this length should not be affected by miner manipulation.

3.4 (HAL-04) MISSING EVENTS EMITTING - INFORMATIONAL

Description:

We observed that some critical functionality are missing emitting any events like `exit` and `exitAll` functions, the governance would probably want to monitor these operations

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider emitting an event when calling `exit` or `exitAll` function.

Listing 4

```
1 event exit(address strategy, uint256 amount);
2 event exitAll(address strategy)
```

Remediation Plan:

SOLVED: Added event emitters for the recommended `exit` and `exitAll` methods. Added additional events where appropriate. Fixed in commit ID: `240a1a261b7f3a9e11031f7a078557073bc7b07d`

3.5 (HAL-05) MISSING RE-ENTRANCY PROTECTION - INFORMATIONAL

Description:

To protect against cross-function reentrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdraw function with a recursive call. OpenZeppelin has it's own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against reentrancy attacks.

Code Location:

Listing 5: OhBank.sol (Lines 197)

```
196 // withdraw an amount of shares for underlying
197     function withdraw(uint256 shares) external override defense {
198         _withdraw(msg.sender, shares);
199     }
200
```

Listing 6: OhCompoundStrategy.sol (Lines 87)

```
86     // withdraw all underlying by redeem all cTokens
87     function withdrawAll() external override onlyBank {
88         uint256 invested = investedBalance();
89         _withdraw(msg.sender, invested);
90     }
91
```

Listing 7: OhAaveV2Strategy.sol (Lines 76)

```
75    /// @notice
76     function withdraw(uint256 amount) external override onlyBank
          returns (uint256) {
77         uint256 withdrawn = _withdraw(msg.sender, amount);
78         return withdrawn;
```

```
79      }
80
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

In the OhBank.sol , OhCompoundStrategy.sol , OhCurve3PoolStrategy.sol and OhAaveV2Strategy.sol contract, function like withdraw() and withdrawAll (), are missing nonReentrant guard. Use the nonReentrant modifier to avoid introducing future vulnerabilities.

Remediation Plan:

RISK ACCEPTED: Oh!Finance team claims that no changes are necessary. The defense modifier prevents re-entrancy attacks. Since only top-level users can call these functions, it is not possible to execute any code on callbacks.

3.6 (HAL-06) IMPRECISION OF A CONSTANT - INFORMATIONAL

Description:

During the audit, It has been observed that integers with scientific notations are directly compared with function arguments.

Code Location:

OhCompoundStrategy.sol Lines #56,100,101
OhCurve3PoolStrategy.sol Lines #120,123,126

Listing 8: OhCurve3PoolStrategy.sol (Lines)

```
120     uint256 supplyShare = amount.mul(1e18).div(invested);
121
122     // find amount to unstake in 3CRV
123     uint256 unstakeAmount = Math.min(staked, supplyShare.mul(
            staked).div(1e18));
124
125     // find amount to redeem in underlying
126     uint256 redeemAmount = Math.min(invested, supplyShare.mul(
            invested).div(1e18));
```

Recommendation:

It is recommended to define precision values as a constant value at the beginning of contract.

Listing 9

```
1 uint constant PRECISION = 1e18;
```

Remediation Plan:

ACKNOWLEDGED: Oh!Finance team claims that no changes are necessary. Declaring constant variables in proxy contracts introduces upgrade risks and usage of the memory stack is cheaper than referencing stored variables.

3.7 (HAL-07) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.

Code Location:

Listing 10: OhBank.sol (Lines 118)

```
118     function virtualPrice() public view override returns (uint256)
            {
119         uint256 totalSupply = totalSupply();
120         uint256 unit = 10**decimals();
121         return totalSupply == 0 ? unit : virtualBalance().mul(unit
                ).div(totalSupply);
122     }
```

Listing 11: OhAaveV2Strategy.sol (Lines 56)

```
56      function investedBalance() public view override returns (
            uint256) {
57          return derivativeBalance();
58      }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider declaring external variables instead of public variables. A best practice is to use external if expecting a function to only be called externally and public if called internally. Public functions are always accessible, but external functions are only available to external callers.

Remediation Plan:

ACKNOWLEDGED: Oh!Finance team claims that functions were intentionally left public. While the Bank and Strategy contracts have upgradeable logic, proxy implementation prevents changing function signatures. Leaving functions marked as public adds the least restrictions for future upgrades. (e.g. We mark virtualPrice() as external and later want to push an upgrade that uses virtualPrice() in a calculation, this cannot be done without duplicating code).

3.8 (HAL-08) LACK OF LIQUIDITY LOSS PROTECTION - INFORMATIONAL

Description:

The exit and exitAll functions allows the bank owner to transfer the deposited amounts to their account. These situations are often enabled because a single bank role, or a liquidity address has access to remove all the TVL (Total Value Locked) through a withdraw or transfer function. While sometimes, the developer or owner does not intend to do this malicious act, the risk still exists if the private key is stolen since there is nothing preventing the key-holder from calling the withdraw.

Code Location:

Listing 12: OhBank.sol (Lines)

```
120     /// @notice Exit and withdraw a given amount from a strategy
121     /// @param strategy The address of the Strategy to exit
122     function exit(address strategy, uint256 amount) external
            override onlyAuthorized {
123         IStrategy(strategy).withdraw(amount);
124     }
125
126     /// @notice Exit and withdraw all underlying from a given
            strategy
127     function exitAll(address strategy) external override
            onlyAuthorized {
128         IStrategy(strategy).withdrawAll();
129     }
```

Recommendation:

Those functions allows the banks of the system to perform withdraw all amounts from strategy addresses. The bank should be limited to the minimum operations possible that allows pool management. Oh!Finance does only use the onlyBank and onlyAuthorized modifier , the bank role check to perform

critical actions such as enabling transfers on the strategies. However, these functionalities should be split between multiple role based users with multi-signature wallets for each one. Also, It is recommended to add timelock or pause/unpause functionality on the withdraw progress. The latency introduced by time locks can act as a preview for how things might work under the unexpected situations.

Remediation Plan

RISK ACCEPTED Oh!Finance team claims that no change are necessary. The `onlyAuthorized` modifier allows either the Manager or Governance address to perform a Strategy exit. Strategy exits do not allow either of these addresses to withdraw user funds, only the withdrawal from a specific Strategy to the Bank address. Funds then sit on the Bank contract and can be withdrawn by burning Bank tokens as usual. The Manager address will always be a contract. During the initial deployment, the Governance address will be set to the deployer, but will then be updated to a DAO Timelock Contract. These functions were written to facilitate normal interactions between the Banks and Strategies, allow any upgrades, and allow any emergency actions to be taken by Governance in the case of an underlying protocol failure.

MANUAL TESTING

During the manual testing multiple questions where considered while evaluation each of the defined functions:

- Can it be re-called changing admin/roles and permissions?
- Can somehow an external controlled contract call again the function during the execution of it? (Re-entrancy)
- Can it be called twice in the same block and cause issues?
- Do we control sensitive or vulnerable parameters?
- Does the function check for boundaries on the parameters and internal values? Bigger than zero or equal? Argument count, array sizes, integer truncation . . .
- Are the function parameters and variables controlled by external contracts?
- Can we re-initialize contracts?
- Can we withdraw more than allowed?

4.1 Testing if contracts could be reinitialized again.

Custom tests are useful for developers to check if functions and permissions work correctly. Furthermore, they are also useful for security auditors to perform security tests behaving like a malicious user. Then, auditors manually manipulated inputs to check the security in the smart contracts.

Contracts are deployed using a proxy, it's essential to test if the initializer modifier exists and prevent deploying the contract again and possibly gain ownership to the contract by attackers.

s OhBank.sol , OhAaveV2Strategy.sol , OhCompoundStrategy.sol and OhCurve3PoolStrategy.sol are using the initializer modifier from OpenZeppelin. Moreover , manually redeploying again will give revert.

Listing 13: OhBank.sol (Lines 43,59)

```
42 /// @notice Initialize the Bank Logic
43     constructor() initializer {
44         assert(registry() == address(0));
45         assert(underlying() == address(0));
46     }
47
48     /// @notice Initialize the Bank Proxy
49     /// @param name_ The name of the Bank Token
50     /// @param symbol_ The symbol of the Bank Token
51     /// @param registry_ Rhe address of the registry
52     /// @param underlying_ Rhe address of the underlying token
           that is deposited
53     /// @dev Should be called when deploying the proxy contract
54     function initializeBank(
55         string memory name_,
56         string memory symbol_,
57         address registry_,
58         address underlying_
59     ) public initializer {
60
```

```
26
27    it('CheckReDeployment', async () => {
28      const {worker} = fixture;
29      const {bank, registry} = worker;
30
31      // try to initialize the bank
32      expect(bank.initializeBank('redeploying Test', 'T', registry.address, usdc.address));
33    });
34
```

Initialization Check

```
(node:858) UnhandledPromiseRejectionWarning: Error: VM Exception while processing transaction: reverted with reason string 'Initializable: contract is already initialize
d'
    at OhBank.initializer (@openzeppelin/contracts-upgradeable/proxy/Initializable.sol:36)
    at OhBank.initializeBank (contracts/bank/OhBank.sol:55)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:97:5)
    at HardhatNode._mineBlockWithPendingTxs (/home/sakr/Work-Halborn/oh-contracts/node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:1261:23)
    at HardhatNode.mineBlock (/home/sakr/Work-Halborn/oh-contracts/node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:384:16)
    at EthModule._sendTransactionAndReturnHash (/home/sakr/Work-Halborn/oh-contracts/node_modules/hardhat/src/internal/hardhat-network/provider/modules/eth.ts:1379:18)
    at HardhatNetworkProvider.request (/home/sakr/Work-Halborn/oh-contracts/node_modules/hardhat/src/internal/hardhat-network/provider/provider.ts:105:18)
    at EthersProviderWrapper.send (/home/sakr/Work-Halborn/oh-contracts/node_modules/@nomiclabs/hardhat-ethers/src/internal/ethers-provider-wrapper.ts:13:20)
(node:858) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async function without a catch block, or
by rejecting a promise which was not handled with .catch(). To terminate the node process on unhandled promise rejection, use the CLI flag `--unhandled-rejections=strict
` (see https://nodejs.org/api/cli.html#cli_unhandled_rejections_mode). (rejection id: 795)
(node:858) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise rejections that are not handled will terminate the Node.js p
rocess with a non-zero exit code.
```

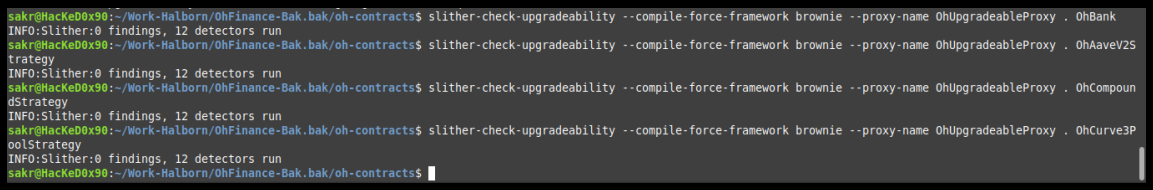
4.2 Testing For Function Clashing.

The DAPP is using proxy to allow quick bug-fixing and adding new features on top of already deployed contracts. it's essential to know in such design it's possible to conceal malicious code that can be very difficult to spot.

Function clashing happens when any function in the Proxy contract whose selector matches with one in the implementation contract will be called directly, completely skipping the implementation code, function selector is the first four bytes of the sha3 of the function signature.

therefore we used slither plugin slither-check-upgradeability to ensure that there is no such clashing between the contracts in scope and the proxy contract.

Function Clashing Test



4.3 Testing For Roles And Privilege.

In this test, it is tried to call critical functions such as exit() and exitAll() to withdraw all funds from strategies. It is observed that access control is correctly implemented and most critical function was either internal or can be only called by Banks or Governors.

```
60
61
62      const  strategy = await usdcBank.strategies(0);
63      await usdcBank.exitAll(strategy);
64
65
```

Exit/ExitAll Function Privilege Check

```
  1) Halborn Exit - ExitAll Function


4 passing (8s)
1 failing

1) Oh! USDC
     Halborn Exit - ExitAll Function:
   Error: VM Exception while processing transaction: reverted with reason string 'Subscriber: Only Authorized'
    at OhBank.onlyAuthorized (contracts/registry/OhSubscriberUpgradeable.sol:18)
    at OhBank.exitAll (contracts/bank/OhBank.sol:146)
    at OhUpgradeableProxy._delegate (@openzeppelin/contracts/proxy/Proxy.sol:31)
    at OhUpgradeableProxy._fallback (@openzeppelin/contracts/proxy/Proxy.sol:56)
    at OhUpgradeableProxy.<fallback> (@openzeppelin/contracts/proxy/Proxy.sol:64)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:97:5)
    at HardhatNode._mineBlockWithPendingTxs (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:1261:23)
    at HardhatNode.mineBlock (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:384:16)
    at EthModule._sendTransactionAndReturnHash (node_modules/hardhat/src/internal/hardhat-network/provider/modules/eth.ts:1379:18)
    at HardhatNetworkProvider.request (node_modules/hardhat/src/internal/hardhat-network/provider/provider.ts:105:18)
    at EthersProviderWrapper.send (node_modules/@nomiclabs/hardhat-ethers/src/internal/ethers-provider-wrapper.ts:13:20)
```

Pause/Unpause Function Privilege Check

Example Code :

```
  it('Halborn Pause - Unpause Function Privilege Check', async () => {
    const {worker} = fixture;
    const {usdcBank, manager} = worker;

    const balance = await usdc.balanceOf(worker.address);
    console.log('Starting Balance is:', formatUnits(balance.toString(), 6));
    await usdc.approve(usdcBank.address, balance);

    await usdcBank.pause();
    await usdcBank.unpause();
  });
```

Test :

```
    1) Halborn Pause - Unpause Function Privilege Check


  4 passing (8s)
  1 failing

  1) Oh! USDC
       Halborn Pause - Unpause Function Privilege Check:
     Error: VM Exception while processing transaction: reverted with reason string 'Subscriber: Only Governance'
      at OhBank.onlyGovernance (contracts/registry/OhSubscriberUpgradeable.sol:24)
      at OhBank.pause (contracts/bank/OhBank.sol:151)
      at OhUpgradeableProxy._delegate (@openzeppelin/contracts/proxy/Proxy.sol:31)
      at OhUpgradeableProxy._fallback (@openzeppelin/contracts/proxy/Proxy.sol:56)
      at OhUpgradeableProxy.<fallback> (@openzeppelin/contracts/proxy/Proxy.sol:64)
      at runMicrotasks (<anonymous>)
      at processTicksAndRejections (internal/process/task_queues.js:97:5)
      at runNextTicks (internal/process/task_queues.js:66:3)
      at listOnTimeout (internal/timers.js:518:9)
      at processTimers (internal/timers.js:492:7)
      at HardhatNode._mineBlockWithPendingTxs (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:1261:23)
      at HardhatNode.mineBlock (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:384:16)
```

Strategy Functions Privilege Check

Example Code :

```
    it('Halborn Strategy Privilege Check', async () => {
    const {worker} = fixture;
    const {manager, usdcBank, usdcCompStrategy} = worker;


    await usdcCompStrategy.withdrawAll();

  });
```

Test :

```
  1) CompoundStrategy
       Halborn Strategy Privilege Check:
     Error: VM Exception while processing transaction: reverted with reason string 'Strategy: Only Bank'
      at OhCompoundStrategy.onlyBank (contracts/strategies/OhStrategy.sol:25)
      at OhCompoundStrategy.withdrawAll (contracts/strategies/compound/OhCompoundStrategy.sol:81)
      at OhUpgradeableProxy._delegate (@openzeppelin/contracts/proxy/Proxy.sol:31)
      at OhUpgradeableProxy._fallback (@openzeppelin/contracts/proxy/Proxy.sol:56)
      at OhUpgradeableProxy.<fallback> (@openzeppelin/contracts/proxy/Proxy.sol:64)
      at runMicrotasks (<anonymous>)
      at processTicksAndRejections (internal/process/task_queues.js:97:5)
      at HardhatNode._mineBlockWithPendingTxs (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:1261:23)
      at HardhatNode.mineBlock (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:384:16)
      at EthModule._sendTransactionAndReturnHash (node_modules/hardhat/src/internal/hardhat-network/provider/modules/eth.ts:1379:18)
      at HardhatNetworkProvider.request (node_modules/hardhat/src/internal/hardhat-network/provider/provider.ts:105:18)
      at EthersProviderWrapper.send (node_modules/@nomiclabs/hardhat-ethers/src/internal/ethers-provider-wrapper.ts:13:20)
```

4.4 Testing For Burning More Tokens Than owned.

In this test, it is tried to burn more tokens than a user owned through the withdraw function, there were no checks to ensure the balance of the owner before calling the _burn method, however the _burn will automatically revert if the amount to be burned is more than his balance.
we still recommend adding a require statement to ensure the user has enough balance before calling the burn function to optimize gas usage.

```
68
69
70      //trying to withdraw more money than the user owns.
71      await usdcBank.withdraw(2233241121311);
72
73
```

Withdraw Testing

```
  1) Halborn Test Withdraw More Than Shares


4 passing (9s)
1 failing

1) Oh! USDC
     Halborn Test Withdraw More Than Shares:
   Error: VM Exception while processing transaction: reverted with reason string 'ERC20: burn amount exceeds balance'
    at OhBank.sub (@openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol:171)
    at OhBank._burn (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol:260)
    at OhBank._withdraw (contracts/bank/OhBank.sol:238)
    at OhBank.withdraw (contracts/bank/OhBank.sol:198)
    at OhUpgradeableProxy._delegate (@openzeppelin/contracts/proxy/Proxy.sol:31)
    at OhUpgradeableProxy._fallback (@openzeppelin/contracts/proxy/Proxy.sol:56)
    at OhUpgradeableProxy.<fallback> (@openzeppelin/contracts/proxy/Proxy.sol:64)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:97:5)
    at runNextTicks (internal/process/task_queues.js:66:3)
    at listOnTimeout (internal/timers.js:518:9)
    at processTimers (internal/timers.js:492:7)
    at HardhatNode._mineBlockWithPendingTxs (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:1261:23)
    at HardhatNode.mineBlock (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:384:16)
```

4.5 Testing Deposit with Signature

During the test, depositWithPermit function is evaluated. The signature is created with another address. To sum up, The manipulation was not successful on the contract.
Test results can be seen from the below.

Example Code :

```
   // sign permit message for usdc
    const balance = await usdc.balanceOf(worker.address);
    const {message, data} = getPermitMessageData(
      'USD Coin',
      '2',
      usdc.address,
      "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB",
      usdcBank.address,
      balance.toString(),
      0,
      Date.now() + 900
    );
    const {v, r, s} = await signMessageData(worker.address, data);

    await usdcBank.depositWithPermit(balance, worker.address, message.deadline, v, r, s);
```

```
1) Oh! USDC
       Halborn Signature Permit Check:
     Error: VM Exception while processing transaction: reverted with reason string 'EIP2612: invalid signature'
      at <UnrecognizedContract>.<unknown> (0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48)
      at <UnrecognizedContract>.<unknown> (0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48)
      at OhBank.depositWithPermit (contracts/bank/OhBank.sol:192)
      at OhUpgradeableProxy._delegate (@openzeppelin/contracts/proxy/Proxy.sol:31)
      at OhUpgradeableProxy._fallback (@openzeppelin/contracts/proxy/Proxy.sol:56)
      at OhUpgradeableProxy.<fallback> (@openzeppelin/contracts/proxy/Proxy.sol:64)
      at HardhatNode._mineBlockWithPendingTxs (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:1261:23)
      at HardhatNode.mineBlock (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:384:16)
      at EthModule._sendTransactionAndReturnHash (node_modules/hardhat/src/internal/hardhat-network/provider/modules/eth.ts:1379:18)
      at HardhatNetworkProvider.request (node_modules/hardhat/src/internal/hardhat-network/provider/provider.ts:105:18)
      at EthersProviderWrapper.send (node_modules/@nomiclabs/hardhat-ethers/src/internal/ethers-provider-wrapper.ts:13:20)
```

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. 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:

```
INFO:Detectors:
OhForum.execute(uint256) (contracts/governance/OhForum.sol#216-230) sends eth to arbitrary user
        Dangerous calls:
        - IGovernor(governance()).executeTransaction{value: proposal.values[i]}(proposal.targets[i],proposal.values[i],proposal.signatures[i],proposal.calldatas[i],pr
sal.eta) (contracts/governance/OhForum.sol#221-227)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
ERC20PermitUpgradeable.__gap (OpenZeppelin/openzeppelin-contracts-upgradeable@3.4.0/contracts/drafts/ERC20PermitUpgradeable.sol#86) shadows:
        - EIP712Upgradeable.__gap (OpenZeppelin/openzeppelin-contracts-upgradeable@3.4.0/contracts/drafts/EIP712Upgradeable.sol#120)
        - ERC20Upgradeable.__gap (OpenZeppelin/openzeppelin-contracts-upgradeable@3.4.0/contracts/token/ERC20/ERC20Upgradeable.sol#312)
        - ContextUpgradeable.__gap (OpenZeppelin/openzeppelin-contracts-upgradeable@3.4.0/contracts/utils/ContextUpgradeable.sol#31)
ERC20Upgradeable.__gap (OpenZeppelin/openzeppelin-contracts-upgradeable@3.4.0/contracts/token/ERC20/ERC20Upgradeable.sol#312) shadows:
        - ContextUpgradeable.__gap (OpenZeppelin/openzeppelin-contracts-upgradeable@3.4.0/contracts/utils/ContextUpgradeable.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
OhBank._deposit(uint256,address,address) (contracts/bank/OhBank.sol#213-228) ignores return value by IERC20(underlying()).transferFrom(sender,address(this),amount) (c
racts/bank/OhBank.sol#225)
OhTimelock.add(address[],uint256[]) (contracts/OhTimelock.sol#60-79) ignores return value by IERC20(token).transferFrom(msg.sender,address(this),totalAmount) (contrac
OhTimelock.sol#78)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
OhPool.pendingSushi(uint256,address) (contracts/OhPool.sol#107-118) performs a multiplication on the result of a division:
        -sushiReward = blocks.mul(tokenPerBlock).mul(pool.allocPoint) / totalAllocPoint (contracts/OhPool.sol#114)
        -accTokenPerShare = accTokenPerShare.add(sushiReward.mul(1e12) / lpSupply) (contracts/OhPool.sol#115)
OhPool.updatePool(uint256) (contracts/OhPool.sol#132-148) performs a multiplication on the result of a division:
        -tokenReward = blocks.mul(tokenPerBlock).mul(pool.allocPoint) / totalAllocPoint (contracts/OhPool.sol#138-139)
        -pool.accTokenPerShare = pool.accTokenPerShare.add((tokenReward.mul(1e12).div(lpSupply))) (contracts/OhPool.sol#140-142)
OhCurve3PoolStrategy._withdraw(address,uint256) (contracts/strategies/curve/OhCurve3PoolStrategy.sol#111-135) performs a multiplication on the result of a division:
        -supplyShare = amount.mul(1e18).div(invested) (contracts/strategies/curve/OhCurve3PoolStrategy.sol#120)
        -unstakeAmount = Math.min(staked,supplyShare.mul(staked).div(1e18)) (contracts/strategies/curve/OhCurve3PoolStrategy.sol#123)
OhCurve3PoolStrategy._withdraw(address,uint256) (contracts/strategies/curve/OhCurve3PoolStrategy.sol#111-135) performs a multiplication on the result of a division:
        -supplyShare = amount.mul(1e18).div(invested) (contracts/strategies/curve/OhCurve3PoolStrategy.sol#120)
        -redeemAmount = Math.min(invested,supplyShare.mul(invested).div(1e18)) (contracts/strategies/curve/OhCurve3PoolStrategy.sol#126)
OhCompoundStrategy._withdraw(address,uint256) (contracts/strategies/compound/OhCompoundStrategy.sol#93-114) performs a multiplication on the result of a division:
        -supplyShare = amount.mul(1e18).div(invested) (contracts/strategies/compound/OhCompoundStrategy.sol#101)
        -redeemAmount = supplyShare.mul(invested).div(1e18) (contracts/strategies/compound/OhCompoundStrategy.sol#102)
```

- The unchecked transfer issue on OhBank.sol file is already reported above

```
INFO:Detectors:
Reentrancy in OhBank._deposit(uint256,address,address) (contracts/bank/OhBank.sol#213-228):
        External calls:
        - IERC20(underlying()).transferFrom(sender,address(this),amount) (contracts/bank/OhBank.sol#225)
        Event emitted after the call(s):
        - Deposit(recipient,amount) (contracts/bank/OhBank.sol#227)
Reentrancy in OhBank._invest(address,uint256) (contracts/bank/OhBank.sol#202-210):
        External calls:
        - TransferHelper.safeTokenTransfer(strategy,underlying(),amount) (contracts/bank/OhBank.sol#205)
        - IStrategy(strategy).invest() (contracts/bank/OhBank.sol#208)
        Event emitted after the call(s):
        - Invest(strategy,amount) (contracts/bank/OhBank.sol#209)
Reentrancy in OhBank._withdraw(address,uint256) (contracts/bank/OhBank.sol#235-252):
        External calls:
        - _withdrawAll() (contracts/bank/OhBank.sol#244)
                - IStrategy(strategies(i)).withdrawAll() (contracts/bank/OhBank.sol#258)
        - _withdrawRemaining(withdrawAmount.sub(balance)) (contracts/bank/OhBank.sol#246)
                - withdrawn = IStrategy(strategies(index)).withdraw(amount) (contracts/bank/OhBank.sol#272)
                - IManager(manager).setWithdrawIndex(index) (contracts/bank/OhBank.sol#287)
        - TransferHelper.safeTokenTransfer(user,underlying(),withdrawAmount) (contracts/bank/OhBank.sol#250)
        Event emitted after the call(s):
        - Withdraw(user,withdrawAmount) (contracts/bank/OhBank.sol#251)
```

- As recommended above we recommend adding the nonReentrant guard to avoid introducing future vulnerabilities.

5.2 AUTOMATED SECURITY SCAN

MYTHX:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit 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 testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Only security-related findings are shown below.

Results:

OhBank Mythx Output

```
Found 1 job(s). Submit? [y/N]: y
  [####################################]  100%
Report for contracts/bank/OhBank.sol
https://dashboard.mythx.io/#/console/analyses/6a916998-d190-48be-82bf-6b174346fa1f
```

Line	SWC Title	Severity	Short Description
38	(SWC-115) Authorization through tx.origin	Low	Use of "tx.origin" as a part of authorization control.
38	(SWC-115) Authorization through tx.origin	Low	Use of tx.origin as a part of authorization control.
43	(SWC-100) Function Default Visibility	Low	Function visibility is not set.
54	(SWC-000) Unknown	Medium	Function could be marked as external.
105	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.
118	(SWC-000) Unknown	Medium	Function could be marked as external.
256	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.
264	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.

OhAaveV2Strategy Mythx Output

```
Report for contracts/strategies/aave/OhAaveV2Strategy.sol
https://dashboard.mythx.io/#/console/analyses/21e708cb-52e9-4bf1-9eef-0f3519343b27
```

Line	SWC Title	Severity	Short Description
24	(SWC-100) Function Default Visibility	Low	Function visibility is not set.
40	(SWC-000) Unknown	Medium	Function could be marked as external.
56	(SWC-000) Unknown	Medium	Function could be marked as external.

OhCompoundStrategy Mythx Output

Report for contracts/strategies/compound/OhCompoundStrategy.sol
https://dashboard.mythx.io/#/console/analyses/d19051de-6419-4f30-990d-088c7b4c3375

Line	SWC Title	Severity	Short Description
22	(SWC-100) Function Default Visibility	Low	Function visibility is not set.
37	(SWC-000) Unknown	Medium	Function could be marked as external.

OhCurve3PoolStrategy Mythx Output

Report for contracts/strategies/curve/OhCurve3PoolStrategy.sol
https://dashboard.mythx.io/#/console/analyses/fe527061-5f2f-43c8-9b75-aa7955c837ae

Line	SWC Title	Severity	Short Description
23	(SWC-100) Function Default Visibility	Low	Function visibility is not set.
40	(SWC-000) Unknown	Medium	Function could be marked as external.

All relevant findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN