// HALBORN

CentaurSwap - Timelock

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **June 27, 2021 - July 5, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/01/2021	Gabi Urrutia
0.5	Document Updates	07/02/2021	Gokberk Gulgun
1.0	Final Document	07/05/2021	Gabi Urrutia
1.1	Remediation Plan	07/08/2021	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
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Centaur Swap uses single-side staking to provide liquidity. In single-side staking, a liquidity provider only needs to stake a single asset rather than in the pair-based format that is seen on popular AMMs, where an LP needs to stake a pair of tokens in equal value to provide liquidity to a pool.

Centaur Swap is designed to allow individual asset pools to trade with each other to maximise liquidity utilisation. This means that once an asset pool is funded, it can effectively be paired against every other existing asset pool.

Centaur Swap engaged Halborn to conduct a security assessment on their Smart contract beginning on June 27th, 2021 and ending July 5th, 2021. The security assessment was scoped to the smart contract provided in the Github repository Centaur Timelock Smart Contract and an audit of the security risk and implications regarding the changes introduced by the development team at Centaur Swap prior to its production release shortly following the assessments deadline.

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.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week 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 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 work as intended.
- Identify potential security issues with the smart contracts.

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 (brownie console and manual deployments on Ganache)
- Manual testing by custom Python scripts.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- 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 contract:
- CentaurFactoryTimeLock.sol

Commit ID: 74d99348e045dee630641c044a1e615de22466df

Fixed Commit ID: 5454babc5f7914c9181a0784beef5efb99d6cb6b

OUT-OF-SCOPE:
External libraries and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	5	2

LIKELIHOOD

IMPACT

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

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - UNDEFINED ROLE ON THE UNLOCK FUNCTION	Low	ACKNOWLEDGED
HAL02 - REDUNDANT CODE IN THE CONDITION STATEMENT	Low	SOLVED - 07/08/2021
HAL03 - LACK OF TIMELOCK SETTER FUNCTION	Low	SOLVED - 07/08/2021
HAL04 - MISSING EVENT HANDLER	Low	FUTURE RELEASE UPDATE
HAL05 - MISSING ADDRESS VALIDATION	Low	SOLVED - 07/08/2021
HAL06 - LACK OF DELAY DEFINITION ON THE CRITICAL FUNCTIONS	Informational	ACKNOWLEDGED
HAL07 - MISSING CONSTANT DEFINITION	Informational	SOLVED - 07/08/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNDEFINED ROLE ON THE UNLOCK FUNCTION - LOW

Description:

In the CentaurFactoryTimeLock.sol contract, the timelock mechanism has been implemented through unlock function. However, this function is authorized through EMERGENCY_MAINTAINER_ROLE. In the other hand, the contract defined TIMELOCK_ADMIN_ROLE role. The TIMELOCK_ADMIN_ROLE role should be authorized by the function(unlock).

Code Location:

CentaurFactoryTimeLock.sol Line #49

Listing 1: CentaurFactoryTimeLock.sol (Lines 49)

```
49      function unlock() external onlyRole(EMERGENCY_MAINTAINER_ROLE)
            {
50          PENDING_UNLOCK = true;
51          UNLOCK_TIMESTAMP = (block.timestamp).add(TIMELOCK_PERIOD);
52      }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

The TIMELOCK_ADMIN_ROLE function should be modifier for the timelock function.

Listing 2: CentaurFactoryTimeLock.sol (Lines 49)

```
49     function unlock() external onlyRole(TIMELOCK_ADMIN_ROLE) {
50         PENDING_UNLOCK = true;
51         UNLOCK_TIMESTAMP = (block.timestamp).add(TIMELOCK_PERIOD);
52     }
```

Remediation Plan:

ACKNOWLEDGED: CentaurSwap Team claims that this is intended behaviour of the function. Only the EMERGENCY_MAINTAINER_ROLE can initiate unlock and emergency withdraw because it will be governed by a 6/6 Multisig.

3.2 (HAL-02) REDUNDANT CODE IN THE CONDITION STATEMENT - LOW

Description:

The conditional statement on CentaurFactoryTimeLock.sol contains the code hasRole(role, address(0)). It is infeasible for msg.sender to ever be equal to address(0). Consider simplifying this to hasRole(role, _msgSender()).

Code Location:

CentaurFactoryTimeLock.sol Line #31

Listing 3: CentaurFactoryTimeLock.sol (Lines 31)

```
30     modifier onlyRole(bytes32 role) {
31         require(hasRole(role, _msgSender()) || hasRole(role,
               address(0)), "CentaurFactoryTimeLock: NO_PERMISSION");
32         _;
33     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to delete hasRole(role, address(0)) conditional statement from the modifier. The sample solution can be seen below.

Listing 4: CentaurFactoryTimeLock.sol (Lines 31)

```
30     modifier onlyRole(bytes32 role) {
```

```
31          require(hasRole(role, _msgSender()), "
                CentaurFactoryTimeLock: NO_PERMISSION");
32          _;
33      }
```

Remediation Plan:

SOLVED: Conditional statement was removed. The CentaurSwap Team updated the relevant contract.

3.3 (HAL-03) LACK OF TIMELOCK SETTER FUNCTION - LOW

Description:

During the tests, It has been observed that TIMELOCK_PERIOD does not have any function for setting new timelock period.

Code Location:

Listing 5: CentaurFactoryTimeLock.sol (Lines 21)

```
21     uint public TIMELOCK_PERIOD = 1 days;
```

Recommendation:

It is recommended to define function for setting new timelock period. Also, this function should have MAXIMUM_DELAY and MINIMUM_DELAY statements on the timelock.

Remediation Plan:

SOLVED: CentaurSwap Team declared TIMELOCK_PERIOD state variable as constant. They do not intend to modify the timelock period.

3.4 (HAL-04) MISSING EVENT HANDLER - LOW

Description:

In the CentaurFactoryTimeLock contract, the functions do not emit event after the progress. Events are a method of informing the transaction initiator about the actions taken by the called function. It logs its emitted parameters in a specific log history, which can be accessed outside of the contract using some filter parameters.

Code Location:

CentaurFactoryTimeLock.sol

Listing 6: Functions (Lines)

```
1     function createPool(address _baseToken, address _oracle, uint
          _liquidityParameter)
2     function addPool(address _pool)
3     function removePool(address _pool)
4     function transferOwnership(address _owner)
5     function setPoolTradeEnabled(address _pool, bool _tradeEnabled
          )
6     function setPoolDepositEnabled(address _pool, bool
          _depositEnabled)
7     function setPoolWithdrawEnabled(address _pool, bool
          _withdrawEnabled)
8     function setPoolLiquidityParameter(address _pool, uint
          _liquidityParameter)
9     function setAllPoolsTradeEnabled(bool _tradeEnabled)
10    function setAllPoolsDepositEnabled(bool _depositEnabled)
11    function setAllPoolsWithdrawEnabled(bool _withdrawEnabled)
12    function emergencyWithdrawFromPool(address _pool, address
          _token, uint _amount, address _to)
13    function setRouterOnlyEOAEnabled(bool _onlyEOAEnabled)
14    function setRouterContractWhitelist(address _address, bool
          _whitelist)
15    function setSettlementDuration(uint _duration)
16    function setPoolFee(uint _poolFee)
```

```
17      function setPoolLogic(address _poolLogic)
18      function setCloneFactory(address _cloneFactory)
19      function setSettlement(address _settlement)
20      function setRouter(address payable _router)
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider as much as possible declaring events at the end of function. Events can be used to detect the end of the operation.

Remediation Plan:

PENDING: CentaurSwap Team will add event in a future release of CentaurFactory instead of CentaurFactoryTimeLock contract.

3.5 (HAL-05) MISSING ADDRESS VALIDATION - LOW

Description:

The CentaurFactoryTimeLock.sol contract has lack a safety check inside constructor and functions. Setters of address type parameters should include a zero-address check. Otherwise, contract functionality may become inaccessible, or tokens could be burnt forever.

Code Location:

CentaurFactoryTimeLock.sol

Listing 7: CentaurFactoryTimeLock.sol (Lines)

```
35     constructor(ICentaurFactory _centaurFactory, address _admin,
           address _normalMaintainer, address _emergencyMaintainer)
           public {
36         centaurFactory = _centaurFactory;
37
38         _setRoleAdmin(TIMELOCK_ADMIN_ROLE, TIMELOCK_ADMIN_ROLE);
39         _setRoleAdmin(NORMAL_MAINTAINER_ROLE, TIMELOCK_ADMIN_ROLE)
               ;
40         _setRoleAdmin(EMERGENCY_MAINTAINER_ROLE,
               TIMELOCK_ADMIN_ROLE);
41
42         _setupRole(TIMELOCK_ADMIN_ROLE, _admin);
43         _setupRole(NORMAL_MAINTAINER_ROLE, _normalMaintainer);
44         _setupRole(EMERGENCY_MAINTAINER_ROLE, _emergencyMaintainer
               );
45
46         PENDING_UNLOCK = false;
47     }
```

Listing 8: CentaurFactoryTimeLock.sol (Lines)

```
35     constructor(ICentaurFactory _centaurFactory, address _admin,
           address     function createPool(address _baseToken, address
```

```
        _oracle, uint _liquidityParameter)
36      function addPool(address _pool)
37      function removePool(address _pool)
38      function transferOwnership(address _owner)
39      function setPoolTradeEnabled(address _pool, bool _tradeEnabled
            )
40      function setPoolDepositEnabled(address _pool, bool
            _depositEnabled)
41      function setPoolWithdrawEnabled(address _pool, bool
            _withdrawEnabled)
42      function setPoolLiquidityParameter(address _pool, uint
            _liquidityParameter)
43      function emergencyWithdrawFromPool(address _pool, address
            _token, uint _amount, address _to)
44      function setRouterContractWhitelist(address _address, bool
            _whitelist)
45      function setPoolLogic(address _poolLogic)
46      function setCloneFactory(address _cloneFactory)
47      function setSettlement(address _settlement)
48      function setRouter(address payable _router)
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Add proper address validation when assigning a value to a variable from user-supplied data. Better yet, address white-listing/black-listing should be implemented in relevant functions if possible.

For example:

Listing 9: Modifier.sol (Lines 2,3,4)

```
1  modifier validAddress(address addr) {
2      require(addr != address(0), "Address cannot be 0x0");
3      require(addr != address(this), "Address cannot be contract");
4      _;
```

```
5   }
```

Remediation Plan:

SOLVED: CentaurSwap Team added Zero-Address check.

3.6 (HAL-06) LACK OF DELAY DEFINITION ON THE CRITICAL FUNCTIONS - INFORMATIONAL

Description:

In the CentaurFactoryTimeLock contracts, the some of the functions do not have timelock. The timelock is a fixed delay time that allows for some reaction time in the event of an unexpected change that is not agreed upon or malicious, and therefore it is possible to unlock the funds and secure them.

Code Location:

CentaurFactoryTimeLock.sol

Listing 10: Functions (Lines)

```
1     function setPoolFee(uint _poolFee) external onlyRole(
          NORMAL_MAINTAINER_ROLE) {
2         centaurFactory.setPoolFee(_poolFee);
3     }
4
5     function setPoolLogic(address _poolLogic) external onlyRole(
          NORMAL_MAINTAINER_ROLE) {
6         centaurFactory.setPoolLogic(_poolLogic);
7     }
8
9     function setPoolLiquidityParameter(address _pool, uint
          _liquidityParameter) public onlyRole(NORMAL_MAINTAINER_ROLE
          ) {
10        centaurFactory.setPoolLiquidityParameter(_pool,
              _liquidityParameter);
11    }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

The timelock should be stated on the critical changes. Whenever the timelock is set by the functions, no one can reduce the waiting time unless using a governance system or an emergency role.

Remediation Plan:

ACKNOWLEDGED: CentaurSwap Team claims that this is intended. Timelock is only in place for emergency withdrawal. The rest of the functions are governed by Multisig confirmations.

3.7 (HAL-07) MISSING CONSTANT DEFINITION - INFORMATIONAL

Description:

State variables should be declared constant to save gas. Without constant definition, the state variable reading progress is performed through the SLOAD operation which costs 200 gas alone.

Code Location:

CentaurFactoryTimeLock.sol Line #21

Listing 11: CentaurFactoryTimeLock.sol (Lines 21)

```
21     uint public TIMELOCK_PERIOD = 1 days;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Add the constant attributes to state variables that never change.

Remediation Plan:

SOLVED: CentaurSwap Team solved in HAL-03 remediation plan.

3.8 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:
Address.isContract(address) (contracts/libraries/Address.sol#26-35) uses assembly
        - INLINE ASM (contracts/libraries/Address.sol#33)
Address._verifyCallResult(bool,bytes,string) (contracts/libraries/Address.sol#171-188) uses assembly
        - INLINE ASM (contracts/libraries/Address.sol#180-183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used in :
        - Version used: ['=0.6.12', '>=0.5.0', '>=0.6.0<0.8.0', '>=0.6.2<0.8.0']
        - =0.6.12 (contracts/CentaurFactoryTimeLock.sol#3)
        - >=0.5.0 (contracts/interfaces/ICentaurFactory.sol#3)
        - >=0.6.0<0.8.0 (contracts/libraries/AccessControl.sol#3)
        - >=0.6.2<0.8.0 (contracts/libraries/Address.sol#3)
        - >=0.6.0<0.8.0 (contracts/libraries/Context.sol#3)
        - >=0.6.0<0.8.0 (contracts/libraries/EnumerableSet.sol#3)
        - >=0.6.0<0.8.0 (contracts/libraries/SafeMath.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version=0.6.12 (contracts/CentaurFactoryTimeLock.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version>=0.5.0 (contracts/interfaces/ICentaurFactory.sol#3) allows old versions
Pragma version>=0.6.0<0.8.0 (contracts/libraries/AccessControl.sol#3) is too complex
Pragma version>=0.6.2<0.8.0 (contracts/libraries/Address.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/libraries/Context.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/libraries/EnumerableSet.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/libraries/SafeMath.sol#3) is too complex
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (contracts/libraries/Address.sol#53-59):
        - (success) = recipient.call{value: amount}() (contracts/libraries/Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (contracts/libraries/Address.sol#114-121):
        - (success,returndata) = target.call{value: value}(data) (contracts/libraries/Address.sol#119)
Low level call in Address.functionStaticCall(address,bytes,string) (contracts/libraries/Address.sol#139-145):
        - (success,returndata) = target.staticcall(data) (contracts/libraries/Address.sol#143)
Low level call in Address.functionDelegateCall(address,bytes,string) (contracts/libraries/Address.sol#163-169):
        - (success,returndata) = target.delegatecall(data) (contracts/libraries/Address.sol#167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
INFO:Detectors:
Parameter CentaurFactoryTimeLock.createPool(address,address,uint256)._baseToken (contracts/CentaurFactoryTimeLock.sol#56) is not in mixedCase
Parameter CentaurFactoryTimeLock.createPool(address,address,uint256)._oracle (contracts/CentaurFactoryTimeLock.sol#56) is not in mixedCase
Parameter CentaurFactoryTimeLock.createPool(address,address,uint256)._liquidityParameter (contracts/CentaurFactoryTimeLock.sol#56) is not in mixedCase
Parameter CentaurFactoryTimeLock.addPool(address)._pool (contracts/CentaurFactoryTimeLock.sol#60) is not in mixedCase
Parameter CentaurFactoryTimeLock.removePool(address)._pool (contracts/CentaurFactoryTimeLock.sol#64) is not in mixedCase
Parameter CentaurFactoryTimeLock.setPoolTradeEnabled(address,bool)._pool (contracts/CentaurFactoryTimeLock.sol#69) is not in mixedCase
Parameter CentaurFactoryTimeLock.setPoolTradeEnabled(address,bool)._tradeEnabled (contracts/CentaurFactoryTimeLock.sol#69) is not in mixedCase
Parameter CentaurFactoryTimeLock.setPoolDepositEnabled(address,bool)._pool (contracts/CentaurFactoryTimeLock.sol#73) is not in mixedCase
Parameter CentaurFactoryTimeLock.setPoolDepositEnabled(address,bool)._depositEnabled (contracts/CentaurFactoryTimeLock.sol#73) is not in mixedCase
Parameter CentaurFactoryTimeLock.setPoolWithdrawEnabled(address,bool)._pool (contracts/CentaurFactoryTimeLock.sol#77) is not in mixedCase
Parameter CentaurFactoryTimeLock.setPoolWithdrawEnabled(address,bool)._withdrawEnabled (contracts/CentaurFactoryTimeLock.sol#77) is not in mixedCase
Parameter CentaurFactoryTimeLock.setPoolLiquidityParameter(address,uint256)._pool (contracts/CentaurFactoryTimeLock.sol#81) is not in mixedCase
Parameter CentaurFactoryTimeLock.setPoolLiquidityParameter(address,uint256)._liquidityParameter (contracts/CentaurFactoryTimeLock.sol#81) is not in mixedCase
Parameter CentaurFactoryTimeLock.setAllPoolsTradeEnabled(bool)._tradeEnabled (contracts/CentaurFactoryTimeLock.sol#85) is not in mixedCase
Parameter CentaurFactoryTimeLock.setAllPoolsDepositEnabled(bool)._depositEnabled (contracts/CentaurFactoryTimeLock.sol#89) is not in mixedCase
Parameter CentaurFactoryTimeLock.setAllPoolsWithdrawEnabled(bool)._withdrawEnabled (contracts/CentaurFactoryTimeLock.sol#93) is not in mixedCase
Parameter CentaurFactoryTimeLock.emergencyWithdrawFromPool(address,address,uint256,address)._pool (contracts/CentaurFactoryTimeLock.sol#97) is not in mixedCase
Parameter CentaurFactoryTimeLock.emergencyWithdrawFromPool(address,address,uint256,address)._token (contracts/CentaurFactoryTimeLock.sol#97) is not in mixedCase
Parameter CentaurFactoryTimeLock.emergencyWithdrawFromPool(address,address,uint256,address)._amount (contracts/CentaurFactoryTimeLock.sol#97) is not in mixedCase
Parameter CentaurFactoryTimeLock.emergencyWithdrawFromPool(address,address,uint256,address)._to (contracts/CentaurFactoryTimeLock.sol#97) is not in mixedCase
Parameter CentaurFactoryTimeLock.setRouterOnlyEOAEnabled(bool)._onlyEOAEnabled (contracts/CentaurFactoryTimeLock.sol#102) is not in mixedCase
Parameter CentaurFactoryTimeLock.setRouterContractWhitelist(address,bool)._address (contracts/CentaurFactoryTimeLock.sol#106) is not in mixedCase
Parameter CentaurFactoryTimeLock.setRouterContractWhitelist(address,bool)._whitelist (contracts/CentaurFactoryTimeLock.sol#106) is not in mixedCase
Parameter CentaurFactoryTimeLock.setSettlementDuration(uint256)._duration (contracts/CentaurFactoryTimeLock.sol#111) is not in mixedCase
Parameter CentaurFactoryTimeLock.setPoolFee(uint256)._poolFee (contracts/CentaurFactoryTimeLock.sol#116) is not in mixedCase
Parameter CentaurFactoryTimeLock.setPoolLogic(address)._poolLogic (contracts/CentaurFactoryTimeLock.sol#120) is not in mixedCase
Parameter CentaurFactoryTimeLock.setCloneFactory(address)._cloneFactory (contracts/CentaurFactoryTimeLock.sol#124) is not in mixedCase
Parameter CentaurFactoryTimeLock.setSettlement(address)._settlement (contracts/CentaurFactoryTimeLock.sol#128) is not in mixedCase
Parameter CentaurFactoryTimeLock.setRouter(address)._router (contracts/CentaurFactoryTimeLock.sol#132) is not in mixedCase
Variable CentaurFactoryTimeLock.PENDING_UNLOCK (contracts/CentaurFactoryTimeLock.sol#19) is not in mixedCase
Variable CentaurFactoryTimeLock.UNLOCK_TIMESTAMP (contracts/CentaurFactoryTimeLock.sol#20) is not in mixedCase
Variable CentaurFactoryTimeLock.TIMELOCK_PERIOD (contracts/CentaurFactoryTimeLock.sol#21) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformity-to-solidity-naming-conventions
INFO:Detectors:
CentaurFactoryTimeLock.TIMELOCK_PERIOD (contracts/CentaurFactoryTimeLock.sol#21) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

The issue was identified in HAL07 - MISSING CONSTANT DEFINITION.

```
INFO:Detectors:
CentaurFactoryTimeLock.TIMELOCK_PERIOD (contracts/CentaurFactoryTimeLock.sol#21) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
setPoolTradeEnabled(address,bool) should be declared external:
        - CentaurFactoryTimeLock.setPoolTradeEnabled(address,bool) (contracts/CentaurFactoryTimeLock.sol#69-71)
setPoolDepositEnabled(address,bool) should be declared external:
        - CentaurFactoryTimeLock.setPoolDepositEnabled(address,bool) (contracts/CentaurFactoryTimeLock.sol#73-75)
setPoolWithdrawEnabled(address,bool) should be declared external:
        - CentaurFactoryTimeLock.setPoolWithdrawEnabled(address,bool) (contracts/CentaurFactoryTimeLock.sol#77-79)
setPoolliquidityParameter(address,uint256) should be declared external:
        - CentaurFactoryTimeLock.setPoolLiquidityParameter(address,uint256) (contracts/CentaurFactoryTimeLock.sol#81-83)
getRoleMemberCount(bytes32) should be declared external:
        - AccessControl.getRoleMemberCount(bytes32) (contracts/libraries/AccessControl.sol#95-97)
getRoleMember(bytes32,uint256) should be declared external:
        - AccessControl.getRoleMember(bytes32,uint256) (contracts/libraries/AccessControl.sol#111-113)
getRoleAdmin(bytes32) should be declared external:
        - AccessControl.getRoleAdmin(bytes32) (contracts/libraries/AccessControl.sol#121-123)
grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (contracts/libraries/AccessControl.sol#135-139)
revokeRole(bytes32,address) should be declared external:
        - AccessControl.revokeRole(bytes32,address) (contracts/libraries/AccessControl.sol#150-154)
renounceRole(bytes32,address) should be declared external:
        - AccessControl.renounceRole(bytes32,address) (contracts/libraries/AccessControl.sol#170-174)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:contracts/CentaurFactoryTimeLock.sol analyzed (7 contracts with 46 detectors), 58 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

3.9 AUTOMATED SECURITY SCAN

MYTHX:

Description:

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:

CentaurFactoryTimelock.sol

No issues were found by MythX.

THANK YOU FOR CHOOSING

// HALBORN