// HALBORN

CentaurSwap Financial Security Audit

Risk Based Assessment

Prepared by: **Halborn**

Date of Engagement: **June 1st - 17th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/09/2021	Nishit Majithia
0.2	Document Update	06/12/2021	Ferran Celades
1.0	Document Draft	06/16/2021	Gabi Urrutia
1.1	Remediation Plan	07/06/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutial	Halborn	Gabi.Urrutia@halborn.com
Nishit Majithia	Halborn	Nishit.Majithia@halborn.com
Ferran Celades	Halborn	Ferran.Celades@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

Centaur Swap engaged Halborn to conduct an economic audit on their platform beginning on June 1st,2021 and ending June 17th, 2021.

The current generation of DeFi has developed organically, without much scrutiny on the stability of financial risk and security. While DeFi welcomes innovation and the advent of new protocols, despite a great effort spent auditing smart contacts to detect and avoid various forms of vulnerabilities, there has been minimal effort to secure entire protocols.

As such, DeFi protocols join the ecosystem, which lead to both exploits against the protocols themselves as well as multi-step attacks that chain a sequence of protocols across multiple platforms. This can be considered an “environment attack” which takes advantage of certain conditions in the “state” of a DeFi platform, such as TVL, that create a financial in-balance that can lead to price or balance manipulation.

Some protocols, such as flash loans, are merely mechanisms that accelerates these attacks. It does so by requiring no collateral (except for the minor gas costs), which is impossible in the traditional finance due to regulations. As such, flash loans democratize the attack, opening an attack strategy to the masses, and can happen anonymous, rapidly, and programmatically by leveraging environmental conditions.

In result, this audit is a financial focused risk evaluation to help calculate the likelihood of a financial loss on the smart contracts developed and deployed by Centaur Swap, and the functions, protocols, and in the DeFi ecosystem.

The main improvements that can be made to reduce the likelihood and impact of a financial risk are:

- Removing emergencyWithdrawFromPool()and emergencyWithdraw() functions or putting some preventive anti-liquidity loss exposure measures
- Consider using Chainlink Oracle as the sole source for price

updates carries risks.

- Integrate third party protection capabilities, such as OpenZeppelin Defender

After implementing the remediation plan, Centaur Swap team includes a Timelock contract which was audited by Halborn as well as the use of multisignature wallets for certain roles. As a result, Liquidity Loss Exposure issue was reduced from CRITICAL to MEDIUM risk level.

1.2 TEST APPROACH & METHODOLOGY

This framework provides a risk based approach to assess the likelihood of a financial security event based on auditing the interactions and inputs around environmental factors of a smart contract or DeFi protocol.

Given the dynamic nature of such an audit, several approaches are combined to perform a holistic assessment of which developers can make a best effort to protect themselves from a revenue impacting event through risk awareness and mitigating factors.

FINANCIAL AUDIT CLASSIFICATIONS The audit approaches security by categorizing risks into several high level classifications, which are then used to align particular vulnerabilities to these categories. The alignment is broken down into the following:

- PROTOCOL BASED RISK FACTORS
- TRANSACTION BASED RISK FACTORS
- INTERNAL BASED RISK FACTORS
- EXTERNAL BASED RISK FACTORS

FINANCIAL AUDIT SECURITY CATEGORIES Within each of these classifications, the security team performed analysis on several categories of vulnerabilities through various qualitative and quantitative assessments.

A qualitative assessment is one in which data describes qualities or characteristics specific to the smart contract environment or code. It is through observation and manual analysis, and frequently appears in narrative form. Qualitative data may be difficult to precisely measure and analyze due to dynamic conditions, or environment variables that are difficult or impossible to predict or detect. The approach allows the auditors to categorize qualitative data to identify themes that correspond with the security research and to perform quantitative analysis.

A quantitative assessment is one in which vulnerabilities or risks can be identified, counted or compared on a numeric scale. For example, it could be the a threshold level on an Oracle price feed, or the amount of liquidity it takes in a Flash Loan to create an attack. This data is usually gathered using tools, or statistical analysis models.

The specific categories that are in scope and tested for each classification are:

PROTOCOL BASED RISK FACTORS

- RISKS DERIVED FROM FLASH LOANS
- LIQUIDITY MANAGEMENT VULNERABILITIES
- FRONT RUNNING AND SLIPPAGE RISKS
- LIQUIDITY LOSS EXPOSURE ASSESSMENT

TRANSACTION BASED RISK FACTORS

- GOVERNANCE ISSUES
- TOKEN INFLATION RISKS
- GAS BASED ISSUES

INTERNAL BASED RISK FACTORS

- TIMELOCK RISKS
- CODE VULNERABILITIES AND BOUNDS CHECKS
- ACCESS CONTROL OR OWNERSHIP RISKS

EXTERNAL BASED RISK FACTORS

- ORACLE ATTACKS
- PRICE FEED MANIPULATION
- FRONT END USER INTERFACE VULNERABILITIES

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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - **8** - HIGH
7 - **6** - MEDIUM
5 - **4** - LOW

3 - 1 - VERY LOW AND INFORMATIONAL

1.3 SCOPE

IN-SCOPE:
The security assessment was scoped to economic attacks on the smart contracts:
- CentaurFactory.sol
- CentaurLPToken.sol
- CentaurPool.sol
- CentaurRouter.sol
- CentaurSettlement.sol
- WheyFarm.sol
- WheyToken.sol
- All smart contracts under helpers, interfaces and libraries folders.

Commit ID: 697348e14359fe49cc84a56c5d7fc54cd9ed5260

OUT-OF-SCOPE:
External libraries.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	1	1	4

LIKELIHOOD

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

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
DERIVED RISKS FROM FLASH LOANS	Informational	PASSED
FRONT RUNNING AND SLIPPAGE RISKS	Informational	PASSED
LIQUIDITY LOSS EXPOSURE RISKS	Critical	PARTIALLY SOLVED
GOVERNANCE RISKS	Informational	PASSED
TOKEN INFLATION RISKS	Informational	PASSED
TIMELOCK RISKS	Informational	PASSED
ORACLE BASED RISKS	Medium	WATCH OVER

FINDINGS & TECH DETAILS

PROTOCOL BASED RISK FACTORS

4.1 (HAL-01) DERIVED RISKS FROM FLASH LOANS ASSESSMENT - INFORMATIONAL

Description:

All the latest attacks started with a Flash loan, and have lately become typical in arbitrage. Flash loans allow an attacker to borrow some funds without any collateral, and the loan is finished and returned in a single transaction. They are executed atomically and without “risk”. No one is able to intercept a flash loan while it is being performed, then it is used for attackers to quickly get liquidity to fund a financial based attack.

For instance, in one of the latest attacks (PancakeBunny) the attacker took 8 different flash loans. Then, the attacker used the liquidity to perform several actions, one of them was used to influence the valuation of the pool (WBNB+BUSDT) swapping 2.32M WBNB for 3.83M BUSDT.
PancakeBunny Incident

Code Location:

Centaur Swap uses a settlement system that prevents the tokens from being swapped on the same transaction. This system would cause any flash loan to revert to its initial state if pending settlements are detected on either of the swapped pools as seen in Listing 1.

Listing 1: CentaurPool.sol

```
function swapFrom(address _sender) external lock onlyRouter
    tradeAllowed returns (uint amount, uint value) {
    uint balance = IERC20(baseToken).balanceOf(address(this));

    require(balance > baseTokenBalance, 'CentaurSwap:
        INSUFFICIENT_SWAP_AMOUNT');

    // Check if has pendingSettlement
    address settlement = ICentaurFactory(factory).settlement();
    require(!ICentaurSettlement(settlement).hasPendingSettlement(
        _sender, address(this)), 'CentaurSwap: PENDING_SETTLEMENT')
        ;

    amount = balance.sub(baseTokenBalance);
    value = getValueFromAmountIn(amount);

    baseTokenBalance = balance;

    emit AmountIn(_sender, amount);

    return (amount, value);
}
```

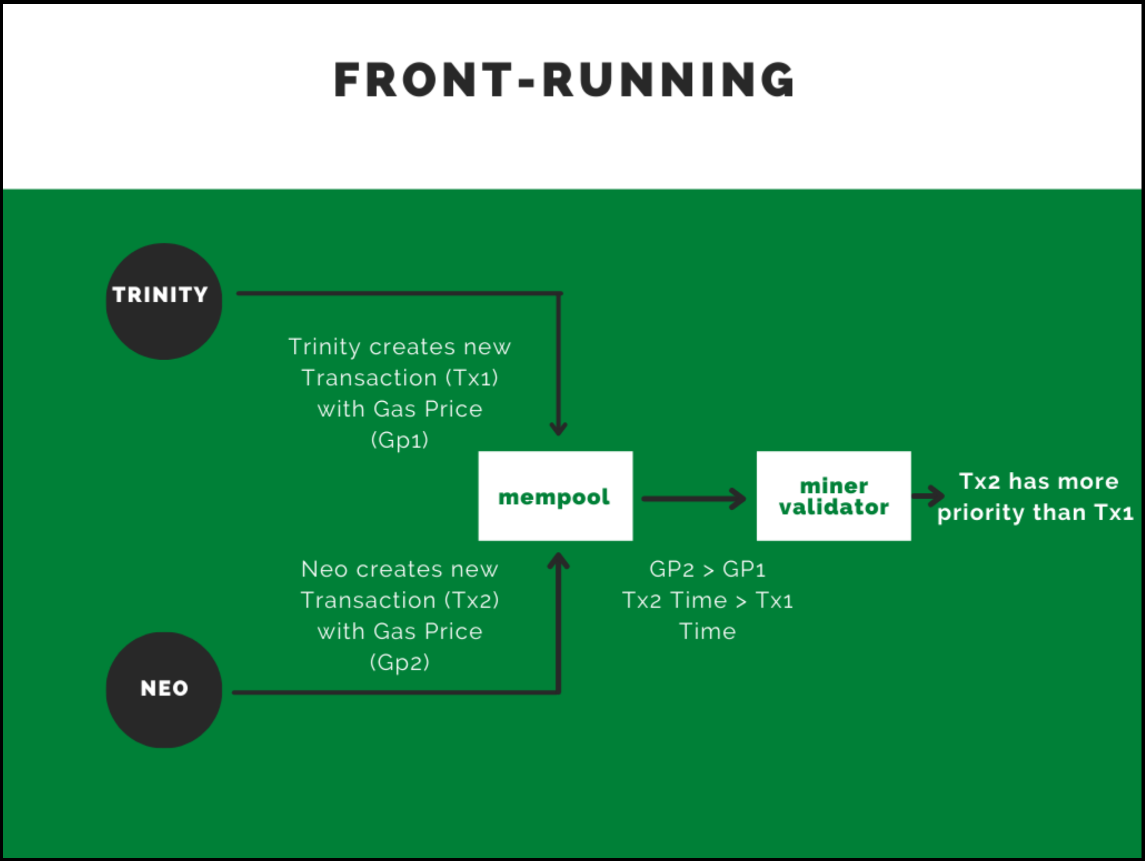
Result:

The assessment is considered successful since flash loans are being prevented by the usage of the settlement system which prevents the swap from fully taking place with the same transaction.

4.2 (HAL-02) FRONT-RUNNING AND SLIPPAGE RISK ASSESSMENT - INFORMATIONAL

Description:

The main issues that make a system vulnerable to front-running attacks are: a lack of transaction confidentiality, and the miner's opportunity to arbitrarily execute transactions that can happen as a consequence of an asynchronous system. To prevent the front-running attack, a reliable method must address these two issues, or the entire structure must be changed to prevent race conditions.


FRONT-RUNNING
TRINITY
Trinity creates new Transaction (Tx1) with Gas Price (Gp1)
mempool
miner validator
Tx2 has more priority than Tx1
Neo creates new Transaction (Tx2) with Gas Price (Gp2)
GP2 > GP1
Tx2 Time > Tx1 Time
NEO

Sandwich Attack Example:

A sandwich attack’s profit is the difference between the tokens spent in the initial buy activity and received in the final sell. The attacker has no control over the initial reserves or the victim’s trade amount; the only variable under his control is the initial amount to trade in the front-run.

1. The first user initiated a normal transaction to get a token swapped. The normal user sets gas price to 10 Gwei.
2. The attacker initiate buy transaction with more gas (20 Gwei). The attacker transaction gains priority under the gas competition mechanism.
3. The Attacker launched another sell transaction at the same moment, this time setting Gas Price to 10 Gwei, which was completed immediately after the normal user’s trade due to the chronological nonce order.
4. Two attack transactions are mixed with one normal transaction, therefore it is named as an sandwich attack.

Result:

Centaur Swap uses a settlement system that prevents the tokens from being swapped on the same transaction. This system would cause any front run attack to be delayed until the settlement is completed. This would prevent sandwich attacks from taking place with the same transaction.

4.3 (HAL-03) LIQUIDITY LOSS EXPOSURE RISK ASSESSMENT - CRITICAL

Description:

Uniswap is a protocol that allows buyers and sellers to swap ERC20 tokens without an exchange or order book. It uses an algorithmic equation that determines the swap rate automatically based on the balances of both tokens, as well as the actual demand for this swapping pair.

For instance, the term known as “rug pull” in De-Fi slang, is a malicious maneuver in the cryptocurrency industry where crypto developers abandon a project and run away with investors’ funds.

Since anyone can spin up a token and smart contract on Ethereum and list it on Uniswap, some developers have come up with unintended liquidity extraction operations. The con begins with minting new tokens, creating Telegram groups to get the buzz going, followed by a Uniswap listing and liquidity injection.

At this point, the original malicious liquidity provider would wait for people to swap their ETH for the newly minted coin, after which the token’s creators would drain the liquidity pool, leaving holders with nothing but a worthless coin.

Example of a Liquidity Loss event:

1. Someone creates a Token Named " XYZ " with total supply of 1 Million and deploys it on the Ethereum network.
2. The rug-puller opens Uniswap and creates a liquidity pool for XYZ / ETH. with 1 million “XYZ” and 100 ETH. (1 ETH = 10,000 XYZ)
3. Then the rug-puller doe the professional marketing for the Token “XYZ” on different platforms that anyone can buy “XYZ” from Uniswap.
4. External users buy “XYZ” from Uniswap Exchange giving ETH or any other ERC-20 token like UNI in Liquidity pool of XYZ / ETH. (Uniswap

converts any Token into ETH automatically).

5. 10 external users buy 30 ETH worth of “XYZ” from Uniswap and no one from them adds XYZ token and ETH to XYZ/ETH pool on uniswap that you created. So it means that the rug-puller has 130 ETH now and 700000 XYZ. (XYZ Token isn’t listed anywhere else so its value is Zero).
6. The rug-puller pulls the trigger and removes the 130 ETH and 700000 XYZ Tokens from Uniswap. The website closes, marketing closes and social networks are removed.

This situation is often enabled because a single owner of a contract, 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:

Furthermore, during code audit some functions were found and considered of critical risk for the liquidity loss exposure assessment. Those functions allows the owners of the system to perform emergency withdraw from any pool with any amount specified. The functions can be seen on Listing 2 and Listing 3.

Listing 2: CentaurFactory.sol

```
function emergencyWithdrawFromPool(address _pool, address _token,
    uint _amount, address _to) external onlyOwner override {
    ICentaurPool(_pool).emergencyWithdraw(_token, _amount, _to);
}
```

Listing 3: CentaurPool.sol

```
function emergencyWithdraw(address _token, uint _amount, address
    _to) external onlyFactory {
    _safeTransfer(_token, _to, _amount);

    emit EmergencyWithdraw(block.timestamp, _token, _amount, _to);
}
```

Result:

The assessment is considered unsuccessful since the functions present on the CentaurFactory and CentaurPool which allows the owner of the pool to perform emergencyWithdraw are considered of high risk and the community should be aware of the existence. It is recommended to completely remove this functionality. Furthermore it is recommended to change the ownership management to a RBAC (Role Based Access Control) system.

The Centaur swap pool does only use the onlyOwner modifier to perform critical actions such as enabling/disabling funds withdrawal on specific pools, enabling transfers and so on. However, this functionality should be split between multiple role based users with multi-signature wallets for each one. For example, for the pausing/unpausing functionality of the entire pools, a Pauser role should be created for example and use its modifier onlyPauser. For the emergencyWithdraw functionality a new role named Emergency should be created independent from the owner and use modifiers such as Emergency isPaused onlyEmergencyOwner and only have this funtionality to take place only once the contract has been paused using the Pauser role. The owner should be limited to the minimum operations possible that allows pool management.

For the liquidity loss exposure protection itself, Centaur Swap uses oracle prices from Chainlink sources and liquidity checks to prevent this type of attacks. However, issues could arise on the data feed causing price manipulation or invalid calculations as explained in the “Price feed - Oracle Risk Assessment”. In order to facilitate the trustworthy of the pool it is recommended to use a service that allows “Proof of Liquidity”.

Proof of Liquidity makes it impossible for scammers to take out their liquidity as they are vested for a period of time to enhance development and innovation on the platform. This is ensured by locking the liquidity of tokens that are pre-launched on their platform. This means that project owners cannot pull the liquidity from their project in a way that cons investors out of their money, or tokens.

As such issues become prevalent across the DeFi system, projects such as Unicrypt, LID Protocol, and Vesta Protocol aim at solving these liquidity problems on Uniswap and similar DEXs in the future.

As it explained before, a rug pull scam mainly arises some minutes, days, or a couple of weeks following the liquidity injection from the hacker. Such a huge sell-off of tokens leads to cascading collapse of projects, whereby the small action of withdrawing liquidity for a profit leads to an eventual collapse of the system.

The most common means that reputable teams use to lock their pooled liquidity and gain additional user confidence and trust are through Unicrypt (https://unicrypt.network/). It's also very easy to verify whether or not liquidity for a particular pair is locked and the date that it is locked to.

Another recommendation is to integrate OpenZepplin Defender platform. With defender, Sentinels are used to automatically monitor and respond to events, functions, and transaction parameters on your smart contracts. With full Autotask integration, you can add circuit breakers or automated actions so your team can respond to attacks within seconds and receive notifications via email, Slack, Telegram, or Discord.

If possible, add to operational processes to research what the coin funds are used for, who the team behind the coins are and the history of trades with this coin.

Remediation Plan:

PARTIALLY SOLVED: Centaur Swap team implemented a Timelock Contract and multi-signature wallets for: Admin, Normal and Emergency roles.

BLOCKCHAIN TRANSACTION BASED RISK FACTORS

5.1 (HAL-04) GOVERNANCE RISK ASSESSMENT - LOW

Description:

In smart contracts which are already running on the blockchain, the upgrading of them is done by a method known as Governance. Governance can be centralized and distributed. Distributed governance allows voters (the ones with the most tokens) to decide actions on smart contracts. If an attacker is able to hijack the governance contract, malicious governance actions provokes a critical risk. Thus, controversial governance proposals can be approved using flash loans.

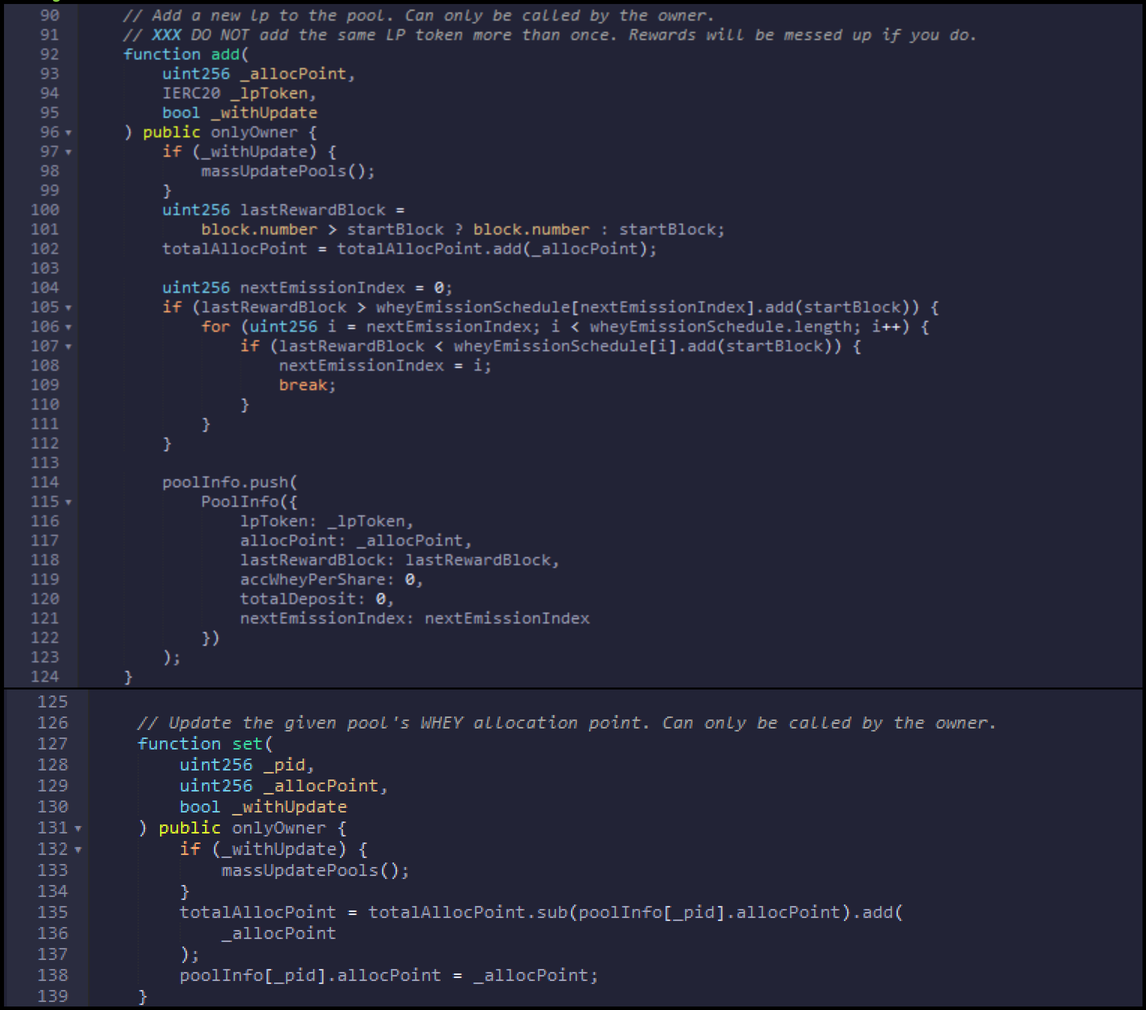
Using flash loans for controlling the Governance

In a distributed Governance, the community should be vigilant. For instance, they can burn all tokens involved in any attack performed by flash loans.

Governance Hijacking

Code Location:

WheyFarm.sol



```
// Add a new lp to the pool. Can only be called by the owner.
// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
function add(
    uint256 _allocPoint,
    IERC20 _lpToken,
    bool _withUpdate
) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardBlock =
        block.number > startBlock ? block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);

    uint256 nextEmissionIndex = 0;
    if (lastRewardBlock > wheyEmissionSchedule[nextEmissionIndex].add(startBlock)) {
        for (uint256 i = nextEmissionIndex; i < wheyEmissionSchedule.length; i++) {
            if (lastRewardBlock < wheyEmissionSchedule[i].add(startBlock)) {
                nextEmissionIndex = i;
                break;
            }
        }
    }

    poolInfo.push(
        PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accWheyPerShare: 0,
            totalDeposit: 0,
            nextEmissionIndex: nextEmissionIndex
        })
    );
}
```

```

// Update the given pool's WHEY allocation point. Can only be called by the owner.
function set(
    uint256 _pid,
    uint256 _allocPoint,
    bool _withUpdate
) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
        _allocPoint
    );
    poolInfo[_pid].allocPoint = _allocPoint;
}
```

Centaur Swap uses a centralized governance to create pools, update the states of the liquidity pools and manage Whey tokens.

Result:

The assessment is considered as passed since governance is centralized. Consider keeping the private key safe or using a multi-signature wallet is enough to prevent the contract from being maliciously updated. Attackers cannot manipulate governance by flash loans.

5.2 (HAL-05) TOKEN INFLATION RISK ASSESSMENT - INFORMATIONAL

Description:

Liquidity providers are responsible for ensuring that liquidity pools are well funded. Ideally, liquidity pools are well funded but there may be under founded liquidity pools (when there is a lack of liquidity by both pairs) and zombie pools.

In zombie pools, there is many tokens from one of the pairs with respect to the other token of the pair. This can cause token inflation and flash loans cause this to happen.

Swapping a huge amount for a token to another can inflate the price of one of the tokens in a pair. Some recent attacks used flash loans to swap from a token to another and unbalanced the token pairs.

Furthermore, not taking into account the possibility of inflationary or deflationary tokens can cause the total amount of the pool to be misinterpreted.

Code Location:

Centaur Swap framework has mechanisms to avoid the token inflation price. First, the owner of CentaurFactory.sol contract is able to lock depositing, withdrawals and trading at the same time and each one separately.

Listing 4: CentaurFactory.sol

```
// Pool Functions
function setPoolTradeEnabled(address _pool, bool _tradeEnabled)
    public onlyOwner override {
    ICentaurPool(_pool).setTradeEnabled(_tradeEnabled);
}

```

```
function setPoolDepositEnabled(address _pool, bool _depositEnabled
    ) public onlyOwner override {
    ICentaurPool(_pool).setDepositEnabled(_depositEnabled);
}

function setPoolWithdrawEnabled(address _pool, bool
    _withdrawEnabled) public onlyOwner override {
    ICentaurPool(_pool).setWithdrawEnabled(_withdrawEnabled);
}
```

In addition, the use of an internal token such as CentaurLPToken. CentaurLPToken.sol manages by minting or burning internal tokens that the pool does not become unbalanced.

Listing 5: CentaurLPToken.sol

```
function _mint(address to, uint value) internal {
    totalSupply = totalSupply.add(value);
    balanceOf[to] = balanceOf[to].add(value);
    emit Transfer(address(0), to, value);
}

function _burn(address from, uint value) internal {
    balanceOf[from] = balanceOf[from].sub(value);
    totalSupply = totalSupply.sub(value);
    emit Transfer(from, address(0), value);
}
```

Furthermore, no issues were found during the manipulation of deflationary or inflationary tokens. As seen on Listing 6 the amount used for the liquidity calculation is taken from the subtraction of the previous baseTokenBalance and the current pool balance of that token. The minted tokens will be taken from that subtraction. With this approach the minted tokens will always take into account the difference between the previous balance and the current one. However, if manual transactions are performed before any liquidity is added those will be accredited to the first one calling the addLiquidity function.

Listing 6: CentaurPool.sol (Lines 101,110)

```
 99 function mint(address to) external lock onlyRouter depositAllowed
       returns (uint liquidity) {
100     uint balance = IERC20(baseToken).balanceOf(address(this));
101     uint amount = balance.sub(baseTokenBalance);
102
103     if (totalSupply == 0) {
104         liquidity = amount.add(baseTokenTargetAmount);
105     } else {
106         liquidity = amount.mul(totalSupply).div(
               baseTokenTargetAmount);
107     }
108
109     require(liquidity > 0, 'CentaurSwap:
           INSUFFICIENT_LIQUIDITY_MINTED');
110     _mint(to, liquidity);
111
112     baseTokenBalance = baseTokenBalance.add(amount);
113     baseTokenTargetAmount = baseTokenTargetAmount.add(amount);
114
115     emit Mint(msg.sender, amount);
116 }
```

Result:

The assessment is considered successful since the pool balance methodology proposed by Centaur Swap team is adequate and prevents manipulating the price of a token and unbalancing a pool.

BLOCKCHAIN SOURCE BASED RISK FACTORS

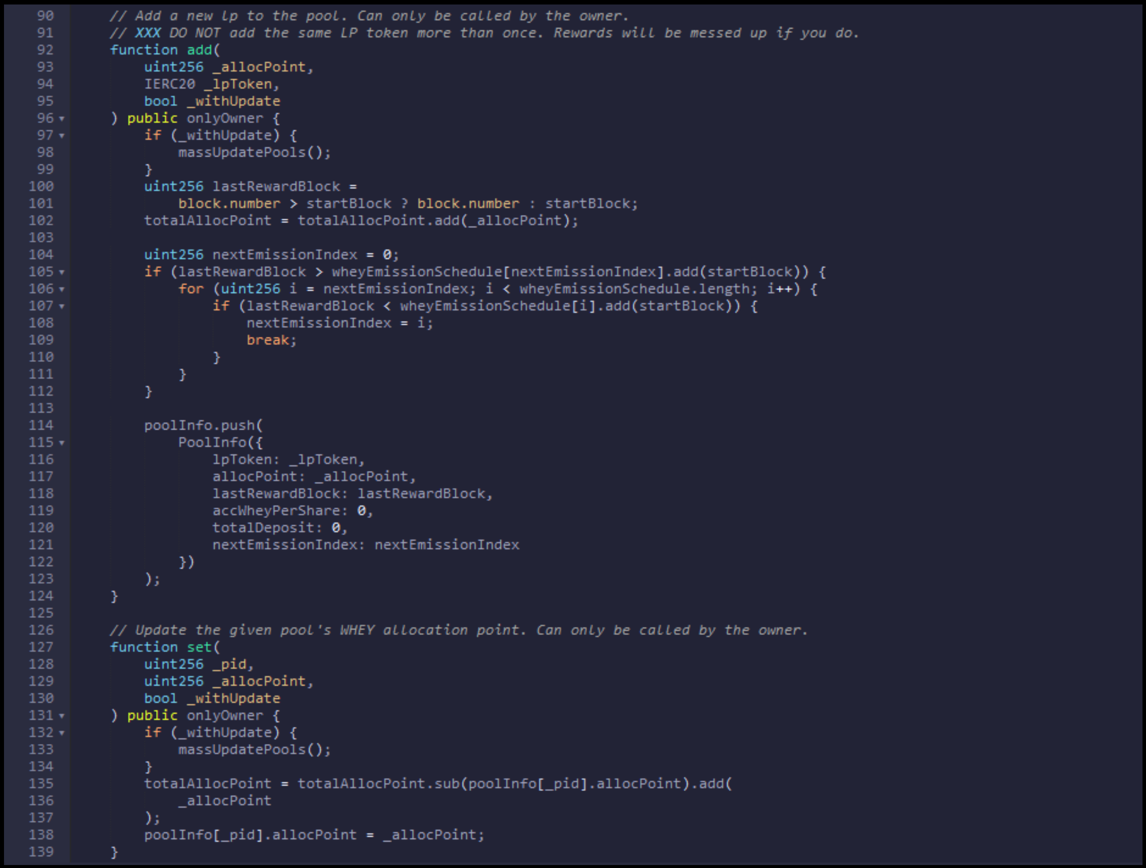
6.1 (HAL-06) TIMELOCK RISK ASSESSMENT - INFORMATIONAL

Description:

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.

Example Codes Without Timelock - Owner Action:

WheyFarm.sol



```
    // Add a new lp to the pool. Can only be called by the owner.
    // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
    function add(
        uint256 _allocPoint,
        IERC20 _lpToken,
        bool _withUpdate
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock =
            block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);

        uint256 nextEmissionIndex = 0;
        if (lastRewardBlock > wheyEmissionSchedule[nextEmissionIndex].add(startBlock)) {
            for (uint256 i = nextEmissionIndex; i < wheyEmissionSchedule.length; i++) {
                if (lastRewardBlock < wheyEmissionSchedule[i].add(startBlock)) {
                    nextEmissionIndex = i;
                    break;
                }
            }
        }

        poolInfo.push(
            PoolInfo({
                lpToken: _lpToken,
                allocPoint: _allocPoint,
                lastRewardBlock: lastRewardBlock,
                accWheyPerShare: 0,
                totalDeposit: 0,
                nextEmissionIndex: nextEmissionIndex
            })
        );
    }

    // Update the given pool's WHEY allocation point. Can only be called by the owner.
    function set(
        uint256 _pid,
        uint256 _allocPoint,
        bool _withUpdate
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
            _allocPoint
        );
        poolInfo[_pid].allocPoint = _allocPoint;
    }
```

Listing 7

```
MakerDAO - 4 hours Timelock
Uniswap - 48 hours Timelock
SushiSwap - 48 hours Timelock
```

Uniswap Timelock
SushiSwap Timelock

Result:

The assessment is considered successful since Centaur Swap does have three minutes settlement set on their smart contracts. The timelock is set by code, once set no one can reduce the waiting time if not allowed using a governance system, which in this case it is centralized using the onlyOnwer modifier.

NON-BLOCKCHAIN SOURCE BASED RISK FACTORS

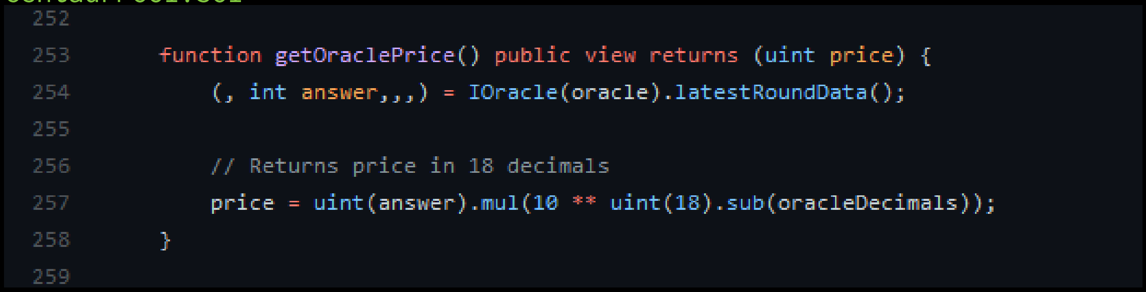
7.1 (HAL-07) PRICE FEED - ORACLE RISK ASSESSMENT - MEDIUM

Description:

An oracle is a technological methodology that converts external data sources into a format that can be used in a blockchain. This allows smart contracts to define state changes and trigger events on a blockchain based on outside external events and interact with the outside world. The problem with oracles is that they create centralized points of trust into systems that are meant to be trustless and decentralized. Since an oracle controls the input data into a smart contract, it therefore controls the operation of the smart contract as it responds to the input data. As an example, If one of oracle feeds incorrect pricing data, arbitrage trading bots will quickly complete huge volume transactions.

Code Location:

CentaurPool.sol



```

    function getOraclePrice() public view returns (uint price) {
        (, int answer,,,) = IOracle(oracle).latestRoundData();

        // Returns price in 18 decimals
        price = uint(answer).mul(10 ** uint(18).sub(oracleDecimals));
    }

```

Result:

Due to oracles are a central point of failure, data sources can go offline and price calculation can cause low liquidity pairs and incorrect values. With the increasing number of oracles, the system design can get complicated and a centralized structure can be constructed. On the other hand, the increase in the number of oracles may cause the system to be open to different attack vectors. As an example solution, The price feed should be obtained from the small group of privileged users. Therefore, an attacker could not directly change price feed via the integration. But, this solution methodology also has disadvantages. The protocol would not able to update the price in the times of protocol. Uniswap introduced new methodology named as Time-Weighted Average Price Oracle on the V2. (https://uniswap.org/docs/v2/core-concepts/oracles/) Time-Weighted Average Price Oracle oracle provides strong protection method to oracle manipulation attacks. However, there are problems in providing a quick feed on this method as well. Although, Chainlink is used as an external oracle in the CentaurSwap contracts, implementing your own oracle will minimize attack surfaces.

THANK YOU FOR CHOOSING

// HALBORN