APY.Finance Financial Security Audit

Risk Based Assessment

Prepared by: **Halborn**

Date of Engagement: **March-June, 2021**

Visit: **Halborn.com**

[redacted]

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/11/2021	Gabi Urrutia
0.2	Document Updates	05/04/2021	Gabi Urrutia
0.3	Document Updates	05/12/2021	Steven Walbroehl
0.4	Document Updates	05/16/2021	Ferran Celades
0.5	Document Updates	05/23/2021	Steven Walbroehl
1.0	Final Version	06/24/2021	Luis Quispe Gonzales
1.1	Remediation Plan	09/30/2021	Luis Quispe Gonzales

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
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

APY.Finance engaged Halborn to conduct an economic audit on their platform beginning on March 15th, 2021 and ending June 23rd, 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 APY Finance, and the functions, protocols, and in the DeFi ecosystem.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by APY.Finance team. The main ones are the following:

- Add a mechanism to restrict the maximum amount of stable coins that can be withdrawn from pools.
- Define an create roles for each kind of operation: day-to-day ops,

interaction with external DeFi protocols and emergency ops.

- Use and configure adequately multisig wallets for each role.
- Implement timelocks for appropriate interactions.
- Add a mutex for unlock function.
- Restrict operations that mint / burn tokens inappropriately.
- Integrate third party protection capabilities, such as OpenZeppelin Defender.
- Revert operations when are outside allowed limits.

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 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 misuse
- Liquidity management vulnerabilities
- Front running and slippage risks
- Liquidity loss exposure

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

EXECUTIVE OVERVIEW

3 - 1 - VERY LOW AND INFORMATIONAL

1.3 SCOPE

IN-SCOPE:

The security assessment was scoped to financial security attacks on the smart contracts:

- AddressRegistryV2.sol
- GovernanceToken.sol
- GovernanceTokenProxy.sol
- Imports.sol
- MetaPoolToken.sol
- MetaPoolTokenProxy.sol
- OracleAdapter.sol
- PoolManager.sol
- PoolManagerProxy.sol
- PoolTokenProxy.sol
- PoolTokenV2.sol
- ProxyConstructorArg.sol
- RewardDistributor.sol
- TVLManager.sol
- All smart contracts under interfaces, periphery and utils folders.

Commit ID: aedab941db048a78e710e5e713cdf5a865f8ea69

OUT-OF-SCOPE:

External libraries.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	5	1	0

LIKELIHOOD

IMPACT

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

The following tables summarize the security issues found:

- First table: Findings, risk level and remediation date.
- Second table: Finding Ids classified according financial security audit criteria.

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01: NO RESTRICTIONS TO COMPLETELY WITHDRAW FROM POOLS TO FUND LP SAFE	Medium	SOLVED - 08/28/2021
HAL-02: UNRESTRICTED CAPACITY TO TRANSFER STABLE COINS FROM LP SAFE TO EXTERNAL ACCOUNTS	High	SOLVED - 09/13/2021
HAL-03: INADEQUATE SEGREGATION OF DUTIES	Medium	SOLVED - 08/28/2021
HAL-04: ABSENCE OF TIMELOCK MECHANISM WHEN UPGRADING SYSTEM LOGIC / PARAMETERS	Medium	PARTIALLY SOLVED
HAL-05: EARLY UNLOCKING OF ORACLE ADAPTER COULD LEAD TO UNFAIR WITHDRAWING	High	SOLVED - 08/28/2021
HAL-06: LOSS OF TOKENS WHEN TRANSACTS WITH SMALL QUANTITIES	Low	SOLVED - 09/30/2021
HAL-07: INSUFFICIENT PROTECTION FOR ORACLE ADAPTER COULD LEAD TO TVL / PRICE MANIPULATION	Medium	SOLVED - 08/28/2021
HAL-08: LACK OF INTERNAL MECHANISMS TO DETECT ABNORMAL VALUES FROM ORACLES	Medium	SOLVED - 08/28/2021

CLASSIFICATION	CATEGORY	FINDINGS
PROTOCOL BASED RISK FACTORS	Risks derived from flash loans misuse	No significant security issues found
PROTOCOL BASED RISK FACTORS	Liquidity management vulnerabilities	HAL-01
PROTOCOL BASED RISK FACTORS	Front running and slippage risks	No significant security issues found
PROTOCOL BASED RISK FACTORS	Liquidity loss exposure	HAL-02
TRANSACTION BASED RISK FACTORS	Governance issues	HAL-03
TRANSACTION BASED RISK FACTORS	Token inflation risks	No significant security issues found
TRANSACTION BASED RISK FACTORS	Gas based issues	No significant security issues found
INTERNAL BASED RISK FACTORS	Timelock risks	HAL-04, HAL-05
INTERNAL BASED RISK FACTORS	Code vulnerabilities and bounds checks	HAL-06
INTERNAL BASED RISK FACTORS	Access control or ownership risks	HAL-07
EXTERNAL BASED RISK FACTORS	Oracle attacks	HAL-08
EXTERNAL BASED RISK FACTORS	Price feed manipulation	No significant security issues found
EXTERNAL BASED RISK FACTORS	Front end user interface vulnerabilities	No significant security issues found

FINDINGS & TECH DETAILS

PROTOCOL BASED RISK FACTORS

4.1 (HAL-01) NO RESTRICTIONS TO COMPLETELY WITHDRAW FROM POOLS TO FUND LP SAFE - MEDIUM

Description:

When owner calls fundLpSafe function from **PoolManager** contract, its _fund function does not verify pools reserve before transferring stable coins to LP Safe, so pools can be completely emptied without restrictions.

This situation could lead legitimate users are not able to redeem its stable coins timely, especially because **pools refill operation must be done manually** by PoolManager owner. The risk level for this finding increases because it also facilitates liquidity loss related attacks (see HAL-02).

Category:

Liquidity management vulnerabilities

Code location:

Listing 1: PoolManager.sol (Lines 147)

```
137 function fundLpSafe(ILpSafeFunder.PoolAmount[] memory poolAmounts)
138         external
139         override
140         onlyOwner
141         nonReentrant
142     {
143         address lpSafeAddress = addressRegistry.lpSafeAddress();
144         require(lpSafeAddress != address(0), "INVALID_LP_SAFE");
145         (PoolTokenV2[] memory pools, uint256[] memory amounts) =
146             _getPoolsAndAmounts(poolAmounts);
147         _fund(lpSafeAddress, pools, amounts);
148         _registerPoolUnderlyers(lpSafeAddress, pools);
149     }
```

Listing 2: PoolManager.sol (Lines 221)

```
202 function _fund(
203         address account,
204         PoolTokenV2[] memory pools,
205         uint256[] memory amounts
206     ) internal {
207         MetaPoolToken mApt = MetaPoolToken(addressRegistry.
            mAptAddress());
208         uint256[] memory mintAmounts = new uint256[](pools.length)
            ;
209         for (uint256 i = 0; i < pools.length; i++) {
210             PoolTokenV2 pool = pools[i];
211             uint256 poolAmount = amounts[i];
212             require(poolAmount > 0, "INVALID_AMOUNT");
213             IDetailedERC20 underlyer = pool.underlyer();
214 
215             uint256 tokenPrice = pool.getUnderlyerPrice();
216             uint8 decimals = underlyer.decimals();
217             uint256 mintAmount =
218                 mApt.calculateMintAmount(poolAmount, tokenPrice,
                    decimals);
219             mintAmounts[i] = mintAmount;
220 
221             underlyer.safeTransferFrom(address(pool), account,
                poolAmount);
222         }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

Implement a security mechanism that automatically calculates the maximum amount of stable coins that can be withdrawn from pools to fund LP Safe without compromising the reserves. Revert the operation if PoolManager owner tries to withdraw more than allowed.

Remediation plan:

SOLVED: Issue fixed in commit 8d4f1d4c4069ced8462ee95ec36d6ef9e12033e0. The fundLp function from **MetaPoolToken** contract automatically calculates the amount of extra reserves and only transfer that extra amount, it no longer takes a parameter to specify any arbitrary amount. On the other hand, the emergencyFundLp function that can transfer from the pools without restriction is protected by **onlyEmergencyRole** modifier.

4.2 (HAL-02) UNRESTRICTED CAPACITY TO TRANSFER STABLE COINS FROM LP SAFE TO EXTERNAL ACCOUNTS - HIGH

Description:

Due to the design of APY.Finance platform, PoolManager owner periodically funds a special Gnosis Safe wallet - called LP Safe - that has unrestricted capacity to **transfer its whole balance to a (potentially malicious) external account**. The risk level for this finding increases because is possible to empty all pools without restrictions, see HAL-01.

Attack scenario:

1. PoolManager owner funds LP Safe with the whole balance from all pools (see HAL-01 finding).
2. If not appropriately configured, LP Safe can use Gnosis Safe's Multisend transaction builder to transfer its whole own balance to a malicious external account.
3. Legitimate users will not be able to redeem its stable coins anymore.

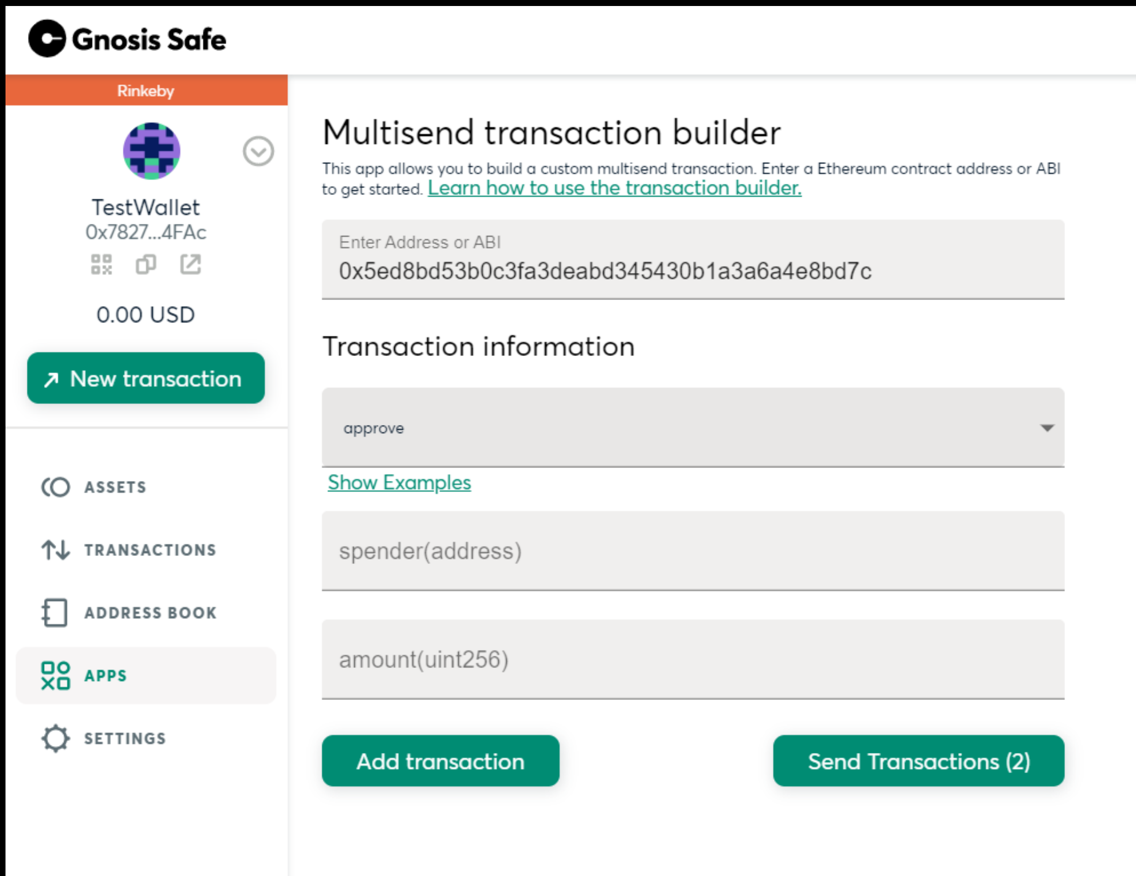
Category:

Liquidity loss exposure

Code location:

Not applicable for a specific smart contract. Some referential images will be shown to highlight Gnosis Safe functionalities that facilitates to transfer (unrestrictedly) stable coins to external accounts.

Referential image of Gnosis Safe's **Multisend transaction builder** that allows to build multiple transactions at once from a smart contract:



Referential image that shows Gnosis Safe capacity to send multiple transactions by signing just once:

Risk Level:

Likelihood - 3

Impact - 5

Recommendations:

Short term security measures to reduce risk level for this finding:

- Solve HAL-01 finding: No restrictions to completely withdraw from pools to fund LP Safe.
- Configure LP Safe's **Owners** section to have a reasonable number of owners and **Policies** section to have an appropriate number of required confirmations to transact.

Long term security measures to reduce risk level for this finding:

- Deploy a proxy smart contract, owned by LP Safe, that interacts with all DeFi protocols used by APY.Finance for yield farming.
- Establish decentralized government to upgrade the interactions with those DeFi protocols or new ones.

Remediation plan:

SOLVED: Issue fixed in commit 35fca40b1d873609a96b09440afc7676666e111d. APY.Finance team implemented a proxy contract called **LpAccount**. This contract holds all the funds and allows the more restrictive admin Safe to register zap contracts. These zap contracts have a consistent interface and define the logic used to interact with external protocols. The **LpAccount** contract can perform a delegate call to these registered zap contracts to deploy, unwind, or swap assets. The configuration of the zap contracts uses hardcoded constants to prevent a malicious user that has compromised the controlling Safe from passing in their own addresses or parameters that could cause a loss of funds.

TRANSACTION BASED RISK FACTORS

5.1 (HAL-03) INADEQUATE SEGREGATION OF DUTIES - MEDIUM

Description:

Due to the design of APY.Finance platform, owners of contracts can carry out different kind of operations:

- **Day-to-day operations:** Fund LP Safe, add asset allocation, etc.
- **Interaction with external DeFi protocols:** Deploy strategy, unwind strategy.
- **Emergency operations:** Set TVL value manually, set TVL aggregator source, etc.

For each kind of operation should exist a different role to reduce the risk of operational mistakes or attacker lateral movement if one of the roles has been already compromised. The risk level for this finding increases because it also facilitates another attacks (see HAL-04, HAL-05 and HAL-07).

Category:

Governance issues

Code location:

Not applicable for a specific smart contract.

Risk Level:

Likelihood - 4
Impact - 3

Recommendations:

Security measures to reduce risk level for this finding:

- Define clearly what functions of APY.Finance protocol should be assigned to each kind of operation.
- Create different roles for each kind of operation: day-to-day ops, interaction with external DeFi protocols and emergency ops.
- Each role must be associated to a multisig wallet and preferably managed by different owners.
- Configure multisig wallets to have a reasonable number of owners and an appropriate number of required confirmations to transact.

Remediation plan:

SOLVED: Issue fixed in commit ab6e8837290b4646827cf5391cdfa75d726c3f1a. APY.Finance team solved this finding by switching from simple contract ownership to role based access control and implementation of this uses the existing OpenZeppelin AccessControl contract. Roles were segregated into three categories, each controlled by a different Gnosis Safe:

- LP Role:
  This role is for day to day management of liquidity and is controlled by the LP Safe.

- Admin Role:
  This role is for configuration that needs to be protected because a malicious actor could use the functions to cause a loss of funds. This is separate from the LP role, so APY.Finance can use a much more restrictive Gnosis Safe. The functions this role protects are not so time-sensitive, so the proper oversight with many signers can be applied. An example is the registration of new LP Account zaps.

- Emergency Role:
  This role is the most restrictive and protects functions that should never be called during normal operation of the system. These functions are powerful and should only be used as a fail-safe in

the event of an emergency. All functions protected by the emergency role are labeled with the prefix “emergency” in addition to the **onlyEmergencyRole** modifier. The big distinction between this and the admin role is that the admin role protects functions that can be used during normal operations.

INTERNAL BASED RISK FACTORS

6.1 (HAL-04) ABSENCE OF TIMELOCK MECHANISM WHEN UPGRADING SYSTEM LOGIC / PARAMETERS - MEDIUM

Description:

APY.Finance protocol does not have a timelock set on their smart contracts, so most owner interactions with the contracts allow him to make large changes to contract configuration (logic / parameters) with no delays or warnings.

Due to operational mistakes or by means of attacks, TVL or prices could be dramatically changed and creates a **great imbalance** on stable coin / APT ratio. This situation would allow users to deposit or withdraw more or less than their fair share, without enough time for APY.Finance team to react against the issue.

Category:

Timelock risks

Code location:

Applicable for all smart contracts within scope. The following are some examples of timelocks used on another protocols:

Listing 3

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

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

Security measures to reduce risk level for this finding:

- Set a timelock by code. Once set, no one can reduce the waiting time unless using a governance system or an emergency role. You can take the examples provided above as a reference.
- For an adequate segregation of duties, create an emergency role (multisig wallet) to manage large changes to contract configuration. This role **must be different** from the one used for day-to-day operations (e.g.: fund LP Safe).
- Configure this multisig wallet to have a reasonable number of owners and an appropriate number of required confirmations to transact.

Remediation plan:

PARTIALLY SOLVED: APY.Finance team implemented part of the suggested remediation to have critical configuration protected by a segregated role (Emergency Role) that uses a more restrictive Safe. This provides more oversight and reduce the likelihood that major changes happens instantly.

6.2 (HAL-05) EARLY UNLOCKING OF ORACLE ADAPTER COULD LEAD TO UNFAIR WITHDRAWING - HIGH

Description:

The unlock function from **OracleAdapter** contract can be called by some permissioned accounts. However, if it is called just after withdrawing LP Safe, TVL value is miscalculated (deployed value is considered twice) and creates a time window where attackers and legitimate users as well can **withdraw much more than allowed** from pools.

Attack scenario:

A step-by-step attack scenario will be described along with screenshots extracted from proof of concept (PoC) script, which is included on security-assessment/tvl_not_updated.js file as an attachment for this report.

1. Attacker deposits its stable coins into a pool.

```
// Attacker deposits its money into a pool
await underlyerToken.connect(attacker).approve(pool.address, deposit);
await pool.connect(attacker).addLiquidity(deposit);

console.log("");
console.log("Attacker has deposited " + argv.deposit + " into " + symbol + " pool ...");
console.logDone();
```

2. PoolManager contract owner withdraws from LP Safe and “mistakenly” unlocks OracleAdapter before TVL value is updated by oracles.

```
// Withdraw from LP Safe to pool
await poolManager.connect(adminSafe).withdrawFromLpSafe([{ poolId: poolId, amount: transferLP }]);

console.logDone();

// Oracle adapter is unlocked "mistakenly" by admin before TVL is updated
await oracleAdapter.connect(adminSafe).unlock();
console.log("Oracle adapter has been unlocked manually by admin");
console.logDone();
```

3. We can see that attacker deposited 1M DAI but is able to withdraw more than 1.39M DAI, which represents a 39% excedent. While TVL value has not been updated by oracles, this vulnerability can be exploited by attackers and legitimate users as well.

```
Attacker has deposited 1000000 into DAI pool ...
√ ... done.
Press ENTER to start the attack!!
Withdrawing 7500000 DAI from LP Safe ...
√ ... done.
Oracle adapter has been unlocked manually by admin
√ ... done.
Attacker has withdrawn from DAI pool and now has 1391884 (39% more)
√ ... done.
```

Category:

Timelock risks

Code location:

Listing 4: OracleAdapter.sol

```
function unlock() external override onlyPermissioned {
        lockFor(0);
}
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendations:

Security measures to reduce risk level for this finding:

- Add a mutex for unlock function. As a reference, you can use a mutex based on roundId value returned by TVL aggregator, so nobody can unlock **OracleAdapter** while roundId has not been incremented with respect to its previous value (before locking). This locking could be reverted on emergency situations using lockFor function.
- For an adequate segregation of duties, create an emergency role (multisig wallet) to unlock OracleAdapter on emergency situations. This role **must be different** from the one used for day-to-day operations (e.g.: fund LP Safe).
- Configure this multisig wallet to have a reasonable number of owners and an appropriate number of required confirmations to transact.

Remediation plan:

SOLVED: Issue fixed in commit 8d4f1d4c4069ced8462ee95ec36d6ef9e12033e0. The unlock function from **OracleAdapter** contract was removed to avoid too early manual unlocking and lockFor function is protected by **onlyContractRole** modifier. On the other hand, the emergencyUnlock function that can unlock contract immediately (in case of emergency) is protected by **onlyEmergencyRole** modifier.

6.3 (HAL-06) LOSS OF TOKENS WHEN TRANSACTS WITH SMALL QUANTITIES - LOW

Description:

When a user calls addLiquidity function from **PoolTokenV2** with a small quantity of stable coins (DAI / USDC / USDT), the **internal function** that calculates the APT tokens to be minted rounds-off the result to zero.

So, the stable coins are added to its corresponding pool but no APT token is minted for the user and generates a very slight imbalance on stable coin / APT ratio. If this issue repeats enough times, the imbalance could lead to APT tokens being mispriced, allowing users to deposit or withdraw more or less than their fair share.

A similar situation occurs when a user calls redeem function from **PoolTokenV2** with a small quantity of APT tokens.

Category:

Code vulnerabilities and bounds checks

Code location:

Listing 5: PoolTokenV2.sol (Lines 237)

```
function addLiquidity(uint256 depositAmount)
        external
        virtual
        override
        nonReentrant
        whenNotPaused
    {
        require(!addLiquidityLock, "LOCKED");
        require(depositAmount > 0, "AMOUNT_INSUFFICIENT");
        require(
```

```
            underlyer.allowance(msg.sender, address(this)) >=
                depositAmount,
            "ALLOWANCE_INSUFFICIENT"
        );
        // solhint-disable-next-line not-rely-on-time
        lastDepositTime[msg.sender] = block.timestamp;

        // calculateMintAmount() is not used because deposit value
        // is needed for the event
        uint256 depositValue = getValueFromUnderlyerAmount(
            depositAmount);
        uint256 poolTotalValue = getPoolTotalValue();
        uint256 mintAmount = _calculateMintAmount(depositValue,
            poolTotalValue);

        _mint(msg.sender, mintAmount);
        underlyer.safeTransferFrom(msg.sender, address(this),
            depositAmount);
```

Listing 6: PoolTokenV2.sol (Lines 280)

```
269 function redeem(uint256 aptAmount)
270         external
271         virtual
272         override
273         nonReentrant
274         whenNotPaused
275     {
276         require(!redeemLock, "LOCKED");
277         require(aptAmount > 0, "AMOUNT_INSUFFICIENT");
278         require(aptAmount <= balanceOf(msg.sender), "
                BALANCE_INSUFFICIENT");
279
280         uint256 redeemUnderlyerAmt = getUnderlyerAmountWithFee(
                aptAmount);
281         require(
282             redeemUnderlyerAmt <= underlyer.balanceOf(address(this
                    )),
283             "RESERVE_INSUFFICIENT"
284         );
285
286         _burn(msg.sender, aptAmount);
287         underlyer.safeTransfer(msg.sender, redeemUnderlyerAmt);
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendations:

Add a **require** function just after calculating APT tokens to be minted (when adding liquidity) or stable coins to be returned (when redeeming). Revert the operation if the result is not greater than zero.

Remediation plan:

SOLVED: After further review, APY.Finance team concluded that there was no way to slowly drain funds from the pool for other users. It could only be used to slowly lose funds for the attacker by depositing tiny amounts and receiving no APT tokens, or redeeming tiny amounts of APT and receiving no funds for it.

Because the additional gas cost of implementing verification exceeded the expected loss to a user from improper use of the feature during normal operation, APY.Finance team decided that it was in the best interest of the user base not to include the verification smart contracts. Instead, what they have done is prevent tiny amounts (< 0.01)from being deposited or redeemed in the front-end UI. In this way, one user cannot accidentally lose a small amount of funds and other users are not affected by the rising cost of gas.

6.4 (HAL-07) INSUFFICIENT PROTECTION FOR ORACLE ADAPTER COULD LEAD TO TVL / PRICE MANIPULATION - MEDIUM

Description:

Currently, the following critical functions on **OracleAdapter** contract are protected by onlyOwner modifier:

- Set TVL value: setTvl
- Set TVL aggregator source: setTvlSource
- Set asset aggregator sources: setAssetSources, setAssetSource

If private key of contract owner is compromised, an attacker could manipulate TVL or prices to create a **great imbalance** on stable coin / APT ratio and **withdraw much more than allowed**.

Attack scenario:

A step-by-step attack scenario will be described along with screenshots extracted from proof of concept (PoC) script, which is included on security-assessment/fake_tvl.js file as an attachment for this report.

The smart contract called FakeAggregator.sol, which is used for this security test, is also included as an attachment.

1. Attacker deposits its stable coins into a pool.

```
// Attacker deposits its money into a pool
await underlyerToken.connect(attacker).approve(pool.address, deposit);
await pool.connect(attacker).addLiquidity(deposit);

console.log("");
console.log("Attacker has deposited " + argv.deposit + " into " + symbol + " pool ...");
console.logDone();
```

2. On the other side, attacker deploys a fake TVL aggregator, which return value can be manipulated anytime by him.

```
console.log("");
console.log("Deploying fake TVL aggregator ...");
const FakeAggregator = await ethers.getContractFactory("FakeAggregator");
const paymentAmount = tokenAmountToBigNumber("1", "18");

// Attacker deploys a fake TVL aggregator
const aggregator = await FakeAggregator.connect(attacker).deploy(
   LINK_ADDRESS,
   paymentAmount,
   100000,
   ZERO_ADDRESS,
   0,
   tokenAmountToBigNumber(1, "20"),
   8,
   "Fake aggregator"
);
```

3. Attacker defines the desired value to be returned by fake TVL aggregator.

```
// Current TVL value returned by oracle
let value = await oracleAdapter.connect(attacker).getTvl();

// New TVL to be returned: TVL * multiplierTVL
value = value.mul(argv.multiplierTVL);
await aggregator.connect(attacker).setValueToSend(value);
```

4. If attacker compromises the private key of OracleAdapter owner, he is able to modify the TVL source to point to fake TVL aggregator previously deployed.

```
/* Attacker sets a fake TVL source
  (Equivalent scenario: Attacker compromises the actual TVL aggregator
  and poisons its return value) */
await oracleAdapter.connect(adminSafe).setTvlSource(aggregator.address);
```

5. Once exploit is launched, we can see that attacker deposited 1M DAI but is able to withdraw more than 5M DAI, which represents a 418% excedent.

```
Attacker has deposited 1000000 into DAI pool ...

√ ... done.

Press ENTER to start the attack!!

Deploying fake TVL aggregator ...
TVL value has been multiplied x10

√ ... done.

Attacker has withdrawn from DAI pool and now has 5189621 (418% more)

√ ... done.
```

Category:

Access control or ownership risks

Code location:

Functions to set **TVL value** and **TVL aggregator source** respectively:

Listing 7: OracleAdapter.sol

```
    function setTvl(uint256 value, uint256 period)
        external
        override
        locked
        onlyOwner
    {
        // We do allow 0 values for submitted values
        submittedTvlValue = Value(value, block.number.add(period))
            ;
    }
```

Listing 8: OracleAdapter.sol

```
function setTvlSource(address source) public onlyOwner {
        require(source.isContract(), "INVALID_SOURCE");
        tvlSource = AggregatorV3Interface(source);
        emit TvlSourceUpdated(source);
    }
```

Functions to set **asset aggregator sources**:

Listing 9: OracleAdapter.sol

```
function setAssetSources(address[] memory assets, address[] memory
    sources)
        public
        onlyOwner
    {
        require(assets.length == sources.length, "
            INCONSISTENT_PARAMS_LENGTH");
        for (uint256 i = 0; i < assets.length; i++) {
            setAssetSource(assets[i], sources[i]);
        }
    }
```

Listing 10: OracleAdapter.sol

```
function setAssetSource(address asset, address source) public
    onlyOwner {
        require(source.isContract(), "INVALID_SOURCE");
        assetSources[asset] = AggregatorV3Interface(source);
        emit AssetSourceUpdated(asset, source);
    }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendations:

Security measures to reduce risk level for this finding:

- For an adequate segregation of duties, create an emergency role (multisig wallet) to manage critical functions on OracleAdapter. This role **must be different** from the one used for day-to-day operations (e.g.: fund LP Safe).

- Configure this multisig wallet to have a reasonable number of owners and an appropriate number of required confirmations to transact.

Remediation plan:

SOLVED: Issue fixed in commit 8d4f1d4c4069ced8462ee95ec36d6ef9e12033e0. APY.Finance team defined the **onlyEmergencyRole** modifier to restrict access to aforementioned functions to the emergency role (multisig wallet).

EXTERNAL BASED RISK FACTORS

7.1 (HAL-08) LACK OF INTERNAL MECHANISMS TO DETECT ABNORMAL VALUES FROM ORACLES - MEDIUM

Description:

The functions on **OracleAdapter** contract that get the TVL and asset price values from oracles (getTvl and getAssetPrice respectively) do not include **previous validations** to verify if received answer is abnormal or has been severely manipulated.

Attack scenario:

1. Attacker deposits its stable coins into a pool.
2. If TVL aggregator is compromised (external attacks or internal fraud), attacker is able to modify the oracles it feeds from to fake ones that inflate the actual TVL value.
3. Attacker and legitimate users as well are able to withdraw much more stable coins than allowed.

Category:

Oracle attacks

Code location:

Listing 11: OracleAdapter.sol (Lines 231)

```
227 function getTvl() external view override unlocked returns (uint256
    ) {
228         if (block.number < submittedTvlValue.periodEnd) {
229             return submittedTvlValue.value;
230         }
231         return _getPriceFromSource(tvlSource);
232     }
```

Listing 12: OracleAdapter.sol (Lines 250)

```
239 function getAssetPrice(address asset)
240         public
241         view
242         override
243         unlocked
244         returns (uint256)
245     {
246         if (block.number < submittedAssetValues[asset].periodEnd)
                {
247             return submittedAssetValues[asset].value;
248         }
249         AggregatorV3Interface source = assetSources[asset];
250         return _getPriceFromSource(source);
251     }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendations:

Calculate TVL value on-chain each time an asset allocation is added or removed. If not possible due to technical restrictions / excessive gas consumption, it is advisable to adopt the following security measures to reduce risk level for this finding:

- Each time getAssetPrice is called, store the latest answer on contract. If a new answer received from price aggregator deviates more than a predefined threshold (e.g: 10%), use instead the latest stored value on contract or fallback to a backup price aggregator.

- Monitor ConfigSet event from **OffchainAggregator** contract to detect timely if there has been logic or oracle (also called transmitters) updates. For this task is possible to integrate OpenZeppelin Defender platform. With Defender, Sentinels are used to automatically monitor and respond to events, functions, and transaction parameters on smart contracts. Also with full Autotask integration, it is feasible to add circuit breakers or automated actions, so APY.Finance team can receive notifications via email, Slack, Telegram or Discord.

- Because of min-max TVL values have been set immutably during **OffchainAggregator** contract deployment, it is advisable to revert operations that could make TVL value be outside the min-max range. It is also possible to ask Chainlink team to upgrade the logic of the contract if necessary.

Remediation plan:

SOLVED: APY.Finance team examined different ways of detecting abnormal oracle values, the most important of which was proper detection of zero values. Zero values require special attention because when a Chainlink aggregator fails, it could return a zero value. However, there are certain valid states of the system in which the TVL can also be zero, such as before the initial deployment of liquidity, or if all liquidity is unwound to deploy a new LP account contract.

To distinguish between valid and invalid zero values, the getTvl function of the **OracleAdapter** contract checks the **totalSupply** of mAPT. When the totalSupply of mAPT is zero, either the liquidity has not been transferred from the LP Account from the pools or all liquidity from the LP Account has been moved back to the pools. A combination of zero mAPT totalSupply and zero TVL indicates a valid zero value. If the totalSupply of mAPT is greater than zero and the oracle still reads a zero TVL, the value should indicate a failure state, and the operation is reverted.

Finally, APY.Finance team decided to detect skew with off-chain monitoring of events that can be responded to using OpenZeppelin Defender.

THANK YOU FOR CHOOSING

HALBORN