APY.Finance

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **June 28th, 2021 - July 16th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/28/2021	Luis Quispe Gonzales
0.2	Document Updates	07/09/2021	Nishit Majithia
0.3	Document Updates	07/13/2021	Luis Quispe Gonzales
1.0	Final Version	07/16/2021	Luis Quispe Gonzales
1.1	Remediation Plan	10/07/2021	Luis Quispe Gonzales

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

APY.Finance engaged Halborn to conduct a security assessment on smart contracts beginning on June 28th, 2021 and ending July 16th, 2021.

The security engineers involved on the audit are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

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

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:

- Restrict minting / burning functions to be called only internally.
- Split privileged address transfer functionality to allow transfer to be completed by recipient.
- Add address validation for user-supplied values in addition to the existing RBAC controls.
- Add events for all relevant operations to help monitor the contracts and detect suspicious behavior.

External threats, such as financial related attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 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.
- Manual testing by custom 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.

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 the following 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.

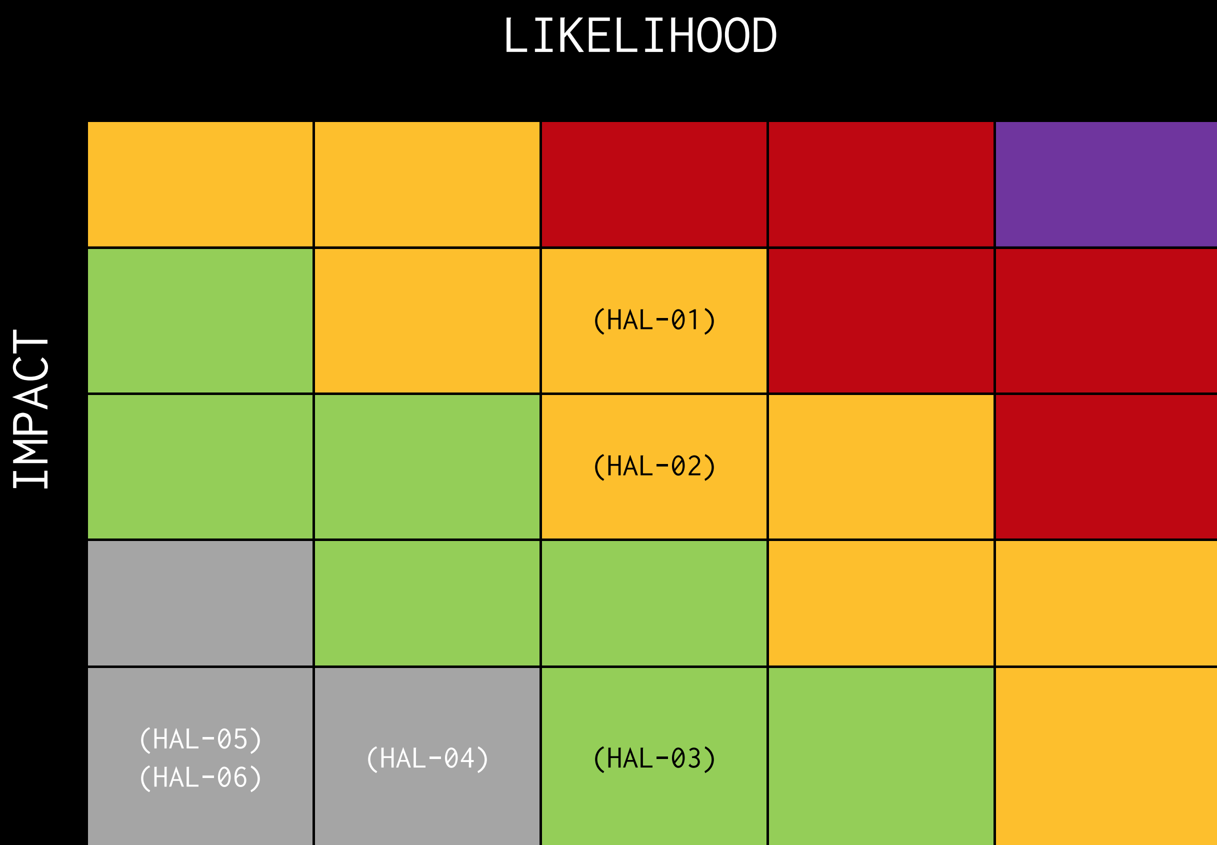
Commit ID: aedab941db048a78e710e5e713cdf5a865f8ea69

OUT-OF-SCOPE:

External libraries and financial related attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	1	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) POSSIBILITY OF MANUAL MINTING / BURNING OF MAPT TOKENS	Medium	SOLVED - 09/10/2021
(HAL-02) PRIVILEGED ADDRESSES CAN BE TRANSFERRED WITHOUT CONFIRMATION	Medium	SOLVED - 10/07/2021
(HAL-03) MISSING ZERO-ADDRESS CHECK	Low	NOT APPLICABLE
(HAL-04) LACK OF EVENTS FOR RELEVANT OPERATIONS	Informational	ACKNOWLEDGED
(HAL-05) POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 09/10/2021
(HAL-06) EXPERIMENTAL FEATURES ENABLED	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) POSSIBILITY OF MANUAL MINTING / BURNING OF MAPT TOKENS - MEDIUM

Description:

Due to the fact mint function from **MetaPoolToken** smart contract is public and restricted just by onlyManager modifier, a malicious owner can change temporarily **PoolManager** address and manually mint / burn mAPT tokens at will.

This situation would allow malicious owner (or users as well) to deposit or withdraw more or less than their fair share.

Code Location:

Attack scenario:

As a matter of example, a step-by-step attack scenario to mint mAPT tokens manually will be described.

Step 1:
Malicious owner (or even an external attacker) calls registerAddress function with the following parameters:

- id: "poolManager"
- _address: 0xABC...DEF (controlled by malicious owner)

Listing 1: AddressRegistryV2.sol (Lines 114)

```
function registerAddress(bytes32 id, address _address) public
    onlyOwner {
    require(_address != address(0), "Invalid address");
    if (_idToAddress[id] == address(0)) {
        // id wasn't registered before, so add it to the list
        _idList.push(id);
    }
```

```
        _idToAddress[id] = _address;
        emit AddressRegistered(id, _address);
    }
```

Step 2:

Malicious owner calls mint function manually from 0xABC...DEF address. Minting operation can be executed only if onlyManager modifier is correctly verified.

Listing 2: MetaPoolToken.sol (Lines 147)

```
143 function mint(address account, uint256 amount)
144     public
145     override
146     nonReentrant
147     onlyManager
148 {
149     require(amount > 0, "INVALID_MINT_AMOUNT");
150     IOracleAdapter oracleAdapter = _getOracleAdapter();
151     oracleAdapter.lock();
152     _mint(account, amount);
153     emit Mint(account, amount);
154 }
```

Step 3:

Automatically onlyManager modifier verifies if 0xABC...DEF address is equal to the value returned by poolManagerAddress function.

Listing 3: MetaPoolToken.sol (Lines 131)

```
129 modifier onlyManager() {
130     require(
131         msg.sender == addressRegistry.poolManagerAddress(),
132         "MANAGER_ONLY"
133     );
134     _;
135 }
```

Step 4:

Because the address of PoolManager has been previously changed in Step 1, poolManagerAddress function will return 0xABC...DEF. This kind of attack allows malicious owner to mint / burn mAPT tokens manually.

Listing 4: AddressRegistryV2.sol (Lines 160)

```
function poolManagerAddress() public view override returns (
    address) {
    return getAddress("poolManager");
}
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

The internal _fund function from **PoolManager** contract should not transfer stablecoins to LP Safe wallet, neither call public mint function from **MetaPoolToken** contract.

Instead, it should call a new external fund function from **MetaPoolToken** contract. Below is a proposed sample code for this function.

Listing 5: Sample code for fund function

```
function fund(PoolTokenV2 pool, address account, uint256 amount)
    external
    nonReentrant
    onlyManager
{
    require(amount > 0, "INVALID_MINT_AMOUNT");
    underlyer.safeTransferFrom(address(pool), account, amount);
    _mint(account, amount);
}
```

On the other hand, _mint function from **MetaPoolToken** contract must be internal. Below is a proposed sample code for this function.

Listing 6: Sample code for mint function

```
function _mint(address account, uint256 amount)
    internal
    override
    nonReentrant
{
    require(amount > 0, "INVALID_MINT_AMOUNT");
    IOracleAdapter oracleAdapter = _getOracleAdapter();
    oracleAdapter.lock();
    super._mint(account, amount);
    emit Mint(account, amount);
}
```

Finally, a similar security mechanism to the one indicated above should be applied to _burn function as well.

Remediation plan:

SOLVED: Issue fixed in commit 42c56dc7bf169224a628364d95906b2f73411516. **PoolManager** and **MetaPoolToken** contracts have been merged and now mint and burn functions are internal.

3.2 (HAL-02) PRIVILEGED ADDRESSES CAN BE TRANSFERRED WITHOUT CONFIRMATION - MEDIUM

Description:

An incorrect use of the function setAdminAddress in contracts can set them to invalid addresses and inadvertently allow unauthorized upgrades of contract logic. The owner of the contracts can change **proxy admin addresses** using the aforementioned function in a single transaction and without confirmation from the new address.

The affected smart contracts are the following:

- PoolTokenV2
- PoolManager
- MetaPoolToken
- AddressRegistryV2

Code Location:

Listing 7: PoolTokenV2.sol

```
function setAdminAddress(address adminAddress) public onlyOwner {
        require(adminAddress != address(0), "INVALID_ADMIN");
        proxyAdmin = adminAddress;
        emit AdminChanged(adminAddress);
}
```

Listing 8: PoolManager.sol

```
function setAdminAddress(address adminAddress) public onlyOwner {
        require(adminAddress != address(0), "INVALID_ADMIN");
        proxyAdmin = adminAddress;
        emit AdminChanged(adminAddress);
}
```

Listing 9: MetaPoolToken.sol

```
function setAdminAddress(address adminAddress) public onlyOwner {
        require(adminAddress != address(0), "INVALID_ADMIN");
        proxyAdmin = adminAddress;
        emit AdminChanged(adminAddress);
}
```

Listing 10: AddressRegistryV2.sol

```
function setAdminAddress(address adminAddress) public onlyOwner {
        require(adminAddress != address(0), "INVALID_ADMIN");
        proxyAdmin = adminAddress;
        emit AdminChanged(adminAddress);
}
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to split **privileged addresses transfer** functionality into setAdminAddress and acceptAdminAddress functions. The latter function allows the transfer to be completed by recipient. Below is a proposed sample code for **acceptAdminAddress** function.

Listing 11: Sample code for acceptAdminAddress function

```
address public proxyAdmin;
address private pendingAdmin;

...

function acceptAdminAddress() external {

   require(msg.sender == pendingAdmin, "Must be proposed admin");

   address oldAdmin = proxyAdmin;
```

```
    proxyAdmin = msg.sender;
    pendingAdmin = address(0);

    emit AdminTransferred(oldAdmin, proxyAdmin);
}
```

Remediation plan:

SOLVED: Issue fixed in commits 42c56dc7bf169224a628364d95906b2f73411516 and 357084f2c3eeed28b80f13d5a8e53f84121726c7.

3.3 (HAL-03) MISSING ZERO-ADDRESS CHECK - LOW

Description:

The setSigner function from **RewardDistributor** contract should perform a zero-address check when receives an address as a user-supplied parameter, despite RBAC controls already implemented (e.g.: onlyOwner modifier).

Code Location:

Listing 12: RewardDistributor.sol (Lines 114)

```
function setSigner(address newSigner) external onlyOwner {
    signer = newSigner;
}
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

Add address validation for user-supplied values in addition to the existing OpenZeppelin RBAC controls.

Remediation plan:

NOT APPLICABLE: APY.Finance team claimed that **RewardDistributor** contract was not in scope for present audit and was reviewed in a previous Halborn audit. Since then, the contract has not changed.

3.4 (HAL-04) LACK OF EVENTS FOR RELEVANT OPERATIONS - INFORMATIONAL

Description:

Several relevant operations do not emit events. As a result, it will be difficult to review the correct behavior of the contracts once deployed.

Relevant operations that would benefit from emitting events include:

- PoolTokenV2.setFeePeriod
- PoolTokenV2.setFeePercentage
- PoolTokenV2.setReservePercentage
- PoolTokenV2.infiniteApprove
- PoolTokenV2.revokeApprove
- PoolManager.fundLpSafe
- PoolManager.withdrawFromLpSafe
- AddressRegistryV2.deleteAddress
- TVLManager.addAssetAllocation
- TVLManager.removeAssetAllocation
- OracleAdapter.setDefaultLockPeriod
- OracleAdapter.setAssetValue
- OracleAdapter.setTvl

Users and/or blockchain monitoring systems are not able to timely detect suspicious behaviors without events.

Code Location:

Below is a sample of relevant functions that do not emit events, which complicates to detect suspicious behavior on smart contracts:

Listing 13: PoolTokenV2.sol

```
function setFeePeriod(uint256 _feePeriod) public onlyOwner {
        feePeriod = _feePeriod;
}
```

Listing 14: PoolManager.sol

```
function fundLpSafe(ILpSafeFunder.PoolAmount[] memory poolAmounts
    )
    external
    override
    onlyOwner
    nonReentrant
{
    address lpSafeAddress = addressRegistry.lpSafeAddress();
    require(lpSafeAddress != address(0), "INVALID_LP_SAFE");
    (PoolTokenV2[] memory pools, uint256[] memory amounts) =
        _getPoolsAndAmounts(poolAmounts);
    _fund(lpSafeAddress, pools, amounts);
    _registerPoolUnderlyers(lpSafeAddress, pools);
}
```

Listing 15: AddressRegistryV2.sol

```
function deleteAddress(bytes32 id) public onlyOwner {
    for (uint256 i = 0; i < _idList.length; i++) {
        if (_idList[i] == id) {
            // copy last element to slot i and shorten array
            _idList[i] = _idList[_idList.length - 1];
            _idList.pop();
            delete _idToAddress[id];
            break;
        }
    }
}
```

Listing 16: TVLManager.sol

```
function addAssetAllocation(
    Data memory data,
    string calldata symbol,
```

```
    uint256 decimals
) external override nonReentrant onlyPermissioned {
    require(!isAssetAllocationRegistered(data), "
        DUPLICATE_DATA_DETECTED");
    bytes32 dataHash = generateDataHash(data);
    _allocationIds.add(dataHash);
    _allocationData[dataHash] = data;
    _allocationSymbols[dataHash] = symbol;
    _allocationDecimals[dataHash] = decimals;
    lockOracleAdapter();
}
```

Listing 17: 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));
}
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Add events for all relevant operations to help monitor the contracts and detect suspicious behavior. A monitoring system that tracks relevant events would allow timely detection of compromised system components.

Remediation plan:

ACKNOWLEDGED: APY.Finance team acknowledged the finding and claimed that they have added events for all functions that have either side-effects or make external calls that could have side-effects (non-view functions).

3.5 (HAL-05) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.
Also, methods do not necessarily have to be public if they are only called within the contract-in such case they should be marked internal.

Code Location:

Below are smart contracts and their corresponding functions affected:

AddressRegistryV2:
getIds, deleteAddress, poolManagerAddress, chainlinkRegistryAddress, daiPoolAddress, usdcPoolAddress, usdtPoolAddress, mAptAddress, lpSafeAddress, oracleAdapterAddress

MetaPoolToken:
mint, burn, calculateMintAmount, calculatePoolAmount, getDeployedValue

PoolTokenV2:
setAddressRegistry, setFeePeriod, setFeePercentage, setReservePercentage, calculateMintAmount, getAPTValue, getUnderlyerAmountFromValue, getReserveTopUpValue

OracleAdapter:
getAssetPrice

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practice, you should use external if you expect that the function will only be called externally and use public if you need to call the function internally. To sum up, all can access to public functions, external functions only can be accessed externally and internal functions can only be called within the contract.

Remediation plan:

SOLVED: Issue fixed in commit 42c56dc7bf169224a628364d95906b2f73411516.

3.6 (HAL-06) EXPERIMENTAL FEATURES ENABLED - INFORMATIONAL

Description:

ABIEncoderV2 is enabled and the use of experimental features could be dangerous on live deployments. The experimental ABI encoder does not handle non-integer values shorter than 32 bytes properly. This applies to bytesNN types, bool, enum and other types when they are part of an array or a struct and encoded directly from storage. This means these storage references have to be used directly inside abi.encode(...) as arguments in external function calls or in event data without prior assignment to a local variable. The types **bytesNN** and **bool** will result in corrupted data while enum might lead to an invalid revert.

Code Location:

Listing 18: Contracts with experimental features enabled

```
AddressRegistryV2
MetaPoolToken
MetaPoolTokenProxy
PoolManager
PoolManagerProxy
PoolTokenProxy
PoolTokenV2
TVLManager
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

When possible, do not use experimental features in the final live deployment. Validate and check that all the conditions above are true for integers and arrays (i.e. all using uint256).

Remediation plan:

ACKNOWLEDGED: APY.Finance team acknowledged the finding and claimed that ABIEncoderV2 is only enabled on contracts where it is strictly necessary.

AUTOMATED TESTING

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
INFO:Detectors:
MetaPoolToken.calculatePoolAmount(uint256,uint256,uint256) (contracts/MetaPoolToken.sol#255-265) performs a multiplication on the result of a division:
        -poolValue = mAptAmount.mul(getTvl()).div(totalSupply()) (contracts/MetaPoolToken.sol#262)
        -poolAmount = poolValue.mul(10 ** decimals).div(tokenPrice) (contracts/MetaPoolToken.sol#263)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

```
INFO:Detectors:
UniswapPeriphery.getPoolBalance(IUniswapV2Pair,uint256).token (contracts/periphery/Uniswap.sol#107) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

- Contract Uniswap.sol has uninitialized local variable IERC20 token, but is false positive and do not impact here.
- Divide before multiply issue in contract MetaPoolToken.sol is false positive and not impacting here.

```
INFO:Detectors:
Reentrancy in MetaPoolToken.burn(address,uint256) (contracts/MetaPoolToken.sol#162-173):
        External calls:
        - oracleAdapter.lock() (contracts/MetaPoolToken.sol#170)
        State variables written after the call(s):
        - _burn(account,amount) (contracts/MetaPoolToken.sol#171)
                - _balances[account] = _balances[account].sub(amount,ERC20: burn amount exceeds balance) (contracts/openzeppelin/contracts-ethereum-packag
/contracts/token/ERC20/ERC20.sol#264)
        - _burn(account,amount) (contracts/MetaPoolToken.sol#171)
                - _totalSupply = _totalSupply.sub(amount) (contracts/openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#265)
Reentrancy in MetaPoolToken.mint(address,uint256) (contracts/MetaPoolToken.sol#143-154):
        External calls:
        - oracleAdapter.lock() (contracts/MetaPoolToken.sol#151)
        State variables written after the call(s):
        - _mint(account,amount) (contracts/MetaPoolToken.sol#152)
                - _balances[account] = _balances[account].add(amount) (contracts/openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#2
4)
        - _mint(account,amount) (contracts/MetaPoolToken.sol#152)
                - _totalSupply = _totalSupply.add(amount) (contracts/openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#243)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in MetaPoolToken.burn(address,uint256) (contracts/MetaPoolToken.sol#162-173):
        External calls:
        - oracleAdapter.lock() (contracts/MetaPoolToken.sol#170)
        Event emitted after the call(s):
        - Burn(account,amount) (contracts/MetaPoolToken.sol#172)
        - Transfer(account,address(0),amount) (contracts/openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#266)
                - _burn(account,amount) (contracts/MetaPoolToken.sol#171)
Reentrancy in MetaPoolToken.mint(address,uint256) (contracts/MetaPoolToken.sol#143-154):
        External calls:
        - oracleAdapter.lock() (contracts/MetaPoolToken.sol#151)
        Event emitted after the call(s):
        - Mint(account,amount) (contracts/MetaPoolToken.sol#153)
        - Transfer(address(0),account,amount) (contracts/openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#245)
                - _mint(account,amount) (contracts/MetaPoolToken.sol#152)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

- Re-entrancy issue is not present and nonReentrant guard is also present to protect against reentrancy attacks.

```
INFO:Detectors:
mint(address,uint256) should be declared external:
        - MetaPoolToken.mint(address,uint256) (contracts/MetaPoolToken.sol#143-154)
burn(address,uint256) should be declared external:
        - MetaPoolToken.burn(address,uint256) (contracts/MetaPoolToken.sol#162-173)
calculateMintAmount(uint256,uint256,uint256) should be declared external:
        - MetaPoolToken.calculateMintAmount(uint256,uint256,uint256) (contracts/MetaPoolToken.sol#210-218)
calculatePoolAmount(uint256,uint256,uint256) should be declared external:
        - MetaPoolToken.calculatePoolAmount(uint256,uint256,uint256) (contracts/MetaPoolToken.sol#255-265)
getDeployedValue(address) should be declared external:
        - MetaPoolToken.getDeployedValue(address) (contracts/MetaPoolToken.sol#272-278)
```

```
INFO:Detectors:
getIds() should be declared external:
        - AddressRegistryV2.getIds() (contracts/AddressRegistryV2.sol#98-100)
deleteAddress(bytes32) should be declared external:
        - AddressRegistryV2.deleteAddress(bytes32) (contracts/AddressRegistryV2.sol#142-152)
poolManagerAddress() should be declared external:
        - AddressRegistryV2.poolManagerAddress() (contracts/AddressRegistryV2.sol#159-161)
chainlinkRegistryAddress() should be declared external:
        - AddressRegistryV2.chainlinkRegistryAddress() (contracts/AddressRegistryV2.sol#179-181)
daiPoolAddress() should be declared external:
        - AddressRegistryV2.daiPoolAddress() (contracts/AddressRegistryV2.sol#188-190)
usdcPoolAddress() should be declared external:
        - AddressRegistryV2.usdcPoolAddress() (contracts/AddressRegistryV2.sol#197-199)
usdtPoolAddress() should be declared external:
        - AddressRegistryV2.usdtPoolAddress() (contracts/AddressRegistryV2.sol#206-208)
mAptAddress() should be declared external:
        - AddressRegistryV2.mAptAddress() (contracts/AddressRegistryV2.sol#210-212)
lpSafeAddress() should be declared external:
        - AddressRegistryV2.lpSafeAddress() (contracts/AddressRegistryV2.sol#217-219)
oracleAdapterAddress() should be declared external:
        - AddressRegistryV2.oracleAdapterAddress() (contracts/AddressRegistryV2.sol#221-223)
```

- Issue regarding misuse of public function has been already mentioned in the above report.

4.2 AUTOMATED SECURITY SCAN

MYTHX:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Only security-related findings are shown below.

Results:

AddressRegistryV2.sol

Report for AddressRegistryV2.sol
https://dashboard.mythx.io/#/console/analyses/64c5f9e8-a328-4f81-a382-167bf4a915bb

Line	SWC Title	Severity	Short Description
98	(SWC-000) Unknown	Medium	Function could be marked as external.
142	(SWC-000) Unknown	Medium	Function could be marked as external.
159	(SWC-000) Unknown	Medium	Function could be marked as external.
179	(SWC-000) Unknown	Medium	Function could be marked as external.
188	(SWC-000) Unknown	Medium	Function could be marked as external.
197	(SWC-000) Unknown	Medium	Function could be marked as external.
206	(SWC-000) Unknown	Medium	Function could be marked as external.
210	(SWC-000) Unknown	Medium	Function could be marked as external.
217	(SWC-000) Unknown	Medium	Function could be marked as external.
221	(SWC-000) Unknown	Medium	Function could be marked as external.

Aave.sol

Report for periphery/Aave.sol
https://dashboard.mythx.io/#/console/analyses/479c667a-b281-416d-9cd6-16dc5dfc5044

Line	SWC Title	Severity	Short Description
64	(SWC-123) Requirement Violation	Low	Requirement violation.
82	(SWC-123) Requirement Violation	Low	Requirement violation.

Curve.sol

Report for periphery/Curve.sol
https://dashboard.mythx.io/#/console/analyses/1d27e384-3714-452d-8019-c33d16c67a76

Line	SWC Title	Severity	Short Description
52	(SWC-123) Requirement Violation	Low	Requirement violation.
88	(SWC-123) Requirement Violation	Low	Requirement violation.
102	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

MetaPoolToken.sol

Report for MetaPoolToken.sol
https://dashboard.mythx.io/#/console/analyses/e664d47f-1d23-4cff-b15c-e14e8dd6f26d

Line	SWC Title	Severity	Short Description
143	(SWC-000) Unknown	Medium	Function could be marked as external.
162	(SWC-000) Unknown	Medium	Function could be marked as external.
192	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
210	(SWC-000) Unknown	Medium	Function could be marked as external.
215	(SWC-101) Integer Overflow and Underflow	High	The arithmetic operator can overflow.
255	(SWC-000) Unknown	Medium	Function could be marked as external.
272	(SWC-000) Unknown	Medium	Function could be marked as external.

Uniswap.sol

Report for periphery/Uniswap.sol
https://dashboard.mythx.io/#/console/analyses/228de8de-9c74-4b2c-bdce-5ab7bbb7cec9

Line	SWC Title	Severity	Short Description
77	(SWC-123) Requirement Violation	Low	Requirement violation.
109	(SWC-123) Requirement Violation	Low	Requirement violation.
115	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
126	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

- Issues regarding 'Integer overflow' is false positive since contract is already using SafeMath from openzeppelin and issue of 'function visibility' has been already raised in the report above.

THANK YOU FOR CHOOSING

HALBORN