Pangolin - Fee Collector v2

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 13th, 2022 - April 18th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/09/2021	Roberto Reigada
0.2	Document Updates	12/10/2021	Roberto Reigada
0.3	Draft Review	12/10/2021	Gabi Urrutia
1.0	Remediation Plan	01/10/2022	Roberto Reigada
1.1	Remediation Plan Review	01/10/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Pangolin engaged Halborn to conduct a security audit on their fee collector smart contract beginning on April 13th, 2022 and ending on April 18th, 2022. The security assessment was scoped to the smart contract provided in the GitHub repository pangolindex/exchange-contracts.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

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

In summary, Halborn did not identify any security risk in the FeeCollector smart contract.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code 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 (Brownie, 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. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

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

1.4 SCOPE

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

- FeeCollector.sol

Commit ID: e28026ce24a3f669923f0357f9b7fbf68755f336

2. EXPLOIT REMEDIATION ANALYSIS

The previous FeeCollector smart contract was exploited due to an issue in the _collectFees() function. The issue was that the _collectFees() function checked the balance of token0 and token1 instead of using the actual amounts returned from burn via the _pullLiquidity() call:

Listing 1: FeeCollector.sol (Lines 951,955)

```
938 function _collectFees(address[] memory liquidityPairs,
939     address outputToken) internal {
940     require(outputToken != address(0), "Output token unspecified")
 ↳ ;
941     for (uint256 i; i < liquidityPairs.length; ++i) {
942         address currentPairAddress = liquidityPairs[i];
943         IPangolinPair currentPair = IPangolinPair(
 ↳ currentPairAddress);
944         uint256 pglBalance = currentPair.balanceOf(address(this));
945         if (pglBalance > 0) {
946             _pullLiquidity(currentPair, pglBalance);
947             address token0 = currentPair.token0();
948             address token1 = currentPair.token1();
949             if (token0 != outputToken) {
950                 _swap(token0, outputToken,
951                     IERC20(token0).balanceOf(address(this)));
952             }
953             if (token1 != outputToken) {
954                 _swap(token1, outputToken,
955                     IERC20(token1).balanceOf(address(this)));
956             }
957         }
958     }
959 }
```

The attacker managed to exploit the smart contract by creating different pairs like AVAX-PGL where the PGL of the pair was already the PGL of another pair, for example, WAVAX-USDC. In this way, the attacker could force the full swap of the other pair through their provided PGL, forcing an altered swap rate and claiming all the swap fees.

This was corrected in the new FeeCollector.

As we can see below, in this new implementation, only the amounts returned by _pullLiquidity() are used in the _swap() calls:

Listing 2: FeeCollector.sol (Lines 216,217,218,219,220,222,225)

```
210 function _convertLiquidity(IPangolinPair[] memory liquidityPairs,
 ↳ address outputToken) private {
211     for (uint256 i; i < liquidityPairs.length; ++i) {
212         IPangolinPair liquidityPair = liquidityPairs[i];
213         uint256 pglBalance = liquidityPair.balanceOf(address(this)
 ↳ );
214         if (pglBalance > 0) {
215             (
216                 address token0,
217                 uint256 token0Pulled,
218                 address token1,
219                 uint256 token1Pulled
220             ) = _pullLiquidity(address(liquidityPair), pglBalance)
 ↳ ;
221             if (token0 != outputToken) {
222                 _swap(token0, outputToken, token0Pulled);
223             }
224             if (token1 != outputToken) {
225                 _swap(token1, outputToken, token1Pulled);
226             }
227         }
228     }
229 }
```

On the other hand, Pangolin team also added a slippage check in the harvest() function:

Listing 3: FeeCollector.sol (Line 252)

```
236 function harvest(
237     IPangolinPair[] calldata liquidityPairs,
238     bool claimMiniChef,
239     uint256 minFinalBalance
240 ) external whenNotPaused onlyRole(HARVEST_ROLE) {
241     address _stakingRewardsRewardToken = stakingRewardsRewardToken
 ↳ ; // Gas savings
242
243     if (liquidityPairs.length > 0) {
```

```
244         _convertLiquidity(liquidityPairs,
 ↳ _stakingRewardsRewardToken);
245     }
246
247     if (claimMiniChef) {
248         IMiniChef(miniChef).harvest(miniChefPoolId, address(this))
 ↳ ;
249     }
250
251     uint256 finalBalance = IERC20(_stakingRewardsRewardToken).
 ↳ balanceOf(address(this));
252     require(finalBalance >= minFinalBalance, "High Slippage");
253
254     uint256 _callIncentive = finalBalance * harvestIncentive /
 ↳ FEE_DENOMINATOR;
255     uint256 _treasuryFee = finalBalance * treasuryFee /
 ↳ FEE_DENOMINATOR;
256     uint256 _totalRewards = finalBalance - _callIncentive -
 ↳ _treasuryFee;
257
258     if (_totalRewards > 0) {
259         address _stakingRewards = stakingRewards;
260         IERC20(_stakingRewardsRewardToken).safeTransfer(
 ↳ _stakingRewards, _totalRewards);
261         IStakingRewards(_stakingRewards).notifyRewardAmount(
 ↳ _totalRewards);
262     }
263     if (_treasuryFee > 0) {
264         IERC20(_stakingRewardsRewardToken).safeTransfer(treasury,
 ↳ _treasuryFee);
265     }
266     if (_callIncentive > 0) {
267         IERC20(_stakingRewardsRewardToken).safeTransfer(msg.sender
 ↳ , _callIncentive);
268     }
269 }
```

This way the harvest() call will be protected against frontrunning.

3. AUTOMATED TESTING

3.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. 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, Slither was run on the all-scoped contracts. 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.

Slither results:

FeeCollector.sol

```
Reentrancy in FeeCollector._convertLiquidity(IPangolinPair[],address) (contracts/fee-collector/FeeCollector.sol#189-206):
        External calls:
        - (token0Pulled,token1Pulled) = _pullLiquidity(address(liquidityPair),pglBalance) (contracts/fee-collector/FeeCollector.sol#197)
                - IERC20(pair).safeTransfer(pair,balance) (contracts/fee-collector/FeeCollector.sol#149)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#93)
                - (amount0,amount1) = IPangolinPair(pair).burn(address(this)) (contracts/fee-collector/FeeCollector.sol#150)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        - _swap(token0,outputToken,token0Pulled) (contracts/fee-collector/FeeCollector.sol#198)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#93)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
                - IERC20(token).safeApprove(ROUTER,type()(uint256).max) (contracts/fee-collector/FeeCollector.sol#173)
                - IPangolinRouter(ROUTER).swapExactTokensForTokens(amount,0,path,address(this),block.timestamp) (contracts/fee-collector/FeeCollector.sol#177-183)
        - _swap(token1,outputToken,token1Pulled) (contracts/fee-collector/FeeCollector.sol#202)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#93)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
                - IERC20(token).safeApprove(ROUTER,type()(uint256).max) (contracts/fee-collector/FeeCollector.sol#173)
                - IPangolinRouter(ROUTER).swapExactTokensForTokens(amount,0,path,address(this),block.timestamp) (contracts/fee-collector/FeeCollector.sol#177-183)
        External calls sending eth:
        - (token0Pulled,token1Pulled) = _pullLiquidity(address(liquidityPair),pglBalance) (contracts/fee-collector/FeeCollector.sol#197)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        - _swap(token0,outputToken,token0Pulled) (contracts/fee-collector/FeeCollector.sol#198)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        - _swap(token1,outputToken,token1Pulled) (contracts/fee-collector/FeeCollector.sol#202)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        State variables written after the call(s):
        - _swap(token1,outputToken,token1Pulled) (contracts/fee-collector/FeeCollector.sol#202)
                - routerApprovals[token] = true (contracts/fee-collector/FeeCollector.sol#174)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

Reentrancy in FeeCollector._swap(address,address,uint256) (contracts/fee-collector/FeeCollector.sol#157-184):
        External calls:
        - IERC20(token).safeApprove(ROUTER,type()(uint256).max) (contracts/fee-collector/FeeCollector.sol#173)
        State variables written after the call(s):
        - routerApprovals[token] = true (contracts/fee-collector/FeeCollector.sol#174)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

FeeCollector._convertLiquidity(IPangolinPair[],address).i (contracts/fee-collector/FeeCollector.sol#190) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

FeeCollector._swap(address,address,uint256) (contracts/fee-collector/FeeCollector.sol#157-184) ignores return value by IPangolinRouter(ROUTER).swapExactTokensForTokens(amount,0,path,address(this),block.timestamp) (contracts/fee-collector
/FeeCollector.sol#177-183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

IMiniChef.poolLength().poolLength (contracts/fee-collector/interfaces/IMiniChef.sol#7) shadows:
        - IMiniChef.poolLength() (contracts/fee-collector/interfaces/IMiniChef.sol#7) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

FeeCollector.constructor(address,address,address,address,address,uint256,address,address,address)._wrappedToken (contracts/fee-collector/FeeCollector.sol#45) lacks a zero-check on :
        - WRAPPED_TOKEN = _wrappedToken (contracts/fee-collector/FeeCollector.sol#55)
FeeCollector.constructor(address,address,address,address,address,uint256,address,address,address)._factory (contracts/fee-collector/FeeCollector.sol#46) lacks a zero-check on :
        - FACTORY = _factory (contracts/fee-collector/FeeCollector.sol#56)
FeeCollector.constructor(address,address,address,address,address,uint256,address,address,address)._router (contracts/fee-collector/FeeCollector.sol#47) lacks a zero-check on :
        - ROUTER = _router (contracts/fee-collector/FeeCollector.sol#57)
FeeCollector.constructor(address,address,address,address,address,uint256,address,address,address)._miniChef (contracts/fee-collector/FeeCollector.sol#49) lacks a zero-check on :
        - miniChef = _miniChef (contracts/fee-collector/FeeCollector.sol#65)
FeeCollector.constructor(address,address,address,address,address,uint256,address,address,address)._treasury (contracts/fee-collector/FeeCollector.sol#51) lacks a zero-check on :
        - treasury = _treasury (contracts/fee-collector/FeeCollector.sol#67)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#213-216)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

EXECUTIVE OVERVIEW

```
Different versions of Solidity is used:
        - Version used: ['^0.8.5', '>=0.5.0', '>=0.6.2', '^0.8.0', '^0.8.1']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/AccessControl.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/IAccessControl.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
        - 0.8.9 (contracts/fee-collector/FeeCollector.sol#3)
        - 0.8.9 (contracts/fee-collector/interfaces/IMiniChef.sol#2)
        - >=0.5.0 (contracts/fee-collector/interfaces/IPangolinPair.sol#1)
        - >=0.6.2 (contracts/fee-collector/interfaces/IPangolinRouter.sol#1)
        - 0.8.9 (contracts/fee-collector/interfaces/IStakingRewards.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

AccessControl._setupRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#185-187) is never used and should be removed
Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#85-87) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-120) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#174-176) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#69-80) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#60-67) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#29-36) is never used and should be removed
Strings.toHexString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#40-51) is never used and should be removed
Strings.toString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#15-35) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/AccessControl.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/IAccessControl.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4) allows old versions
Pragma version0.8.9 (contracts/fee-collector/FeeCollector.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/fee-collector/interfaces/IMiniChef.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version>=0.5.0 (contracts/fee-collector/interfaces/IPangolinPair.sol#1) allows old versions
Pragma version>=0.6.2 (contracts/fee-collector/interfaces/IPangolinRouter.sol#1) allows old versions
Pragma version0.8.9 (contracts/fee-collector/interfaces/IStakingRewards.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter FeeCollector.setRewardsContract(address)._stakingRewards (contracts/fee-collector/FeeCollector.sol#78) is not in mixedCase
Parameter FeeCollector.setHarvestIncentive(uint256)._harvestIncentive (contracts/fee-collector/FeeCollector.sol#88) is not in mixedCase
Parameter FeeCollector.setTreasuryFee(uint256)._treasuryFee (contracts/fee-collector/FeeCollector.sol#106) is not in mixedCase
Parameter FeeCollector.setTreasury(address)._treasury (contracts/fee-collector/FeeCollector.sol#113) is not in mixedCase
Parameter FeeCollector.setMiniChef(address)._miniChef (contracts/fee-collector/FeeCollector.sol#120) is not in mixedCase
Parameter FeeCollector.setMiniChefPool(uint256)._pid (contracts/fee-collector/FeeCollector.sol#127) is not in mixedCase
Parameter FeeCollector.setRewardsDuration(uint256)._rewardsDuration (contracts/fee-collector/FeeCollector.sol#133) is not in mixedCase
Parameter FeeCollector.transferStakingOwnership(address)._newOwner (contracts/fee-collector/FeeCollector.sol#139) is not in mixedCase
Variable FeeCollector.FACTORY (contracts/fee-collector/FeeCollector.sol#20) is not in mixedCase
Variable FeeCollector.ROUTER (contracts/fee-collector/FeeCollector.sol#21) is not in mixedCase
Variable FeeCollector.WRAPPED_TOKEN (contracts/fee-collector/FeeCollector.sol#22) is not in mixedCase
Function IPangolinPair.DOMAIN_SEPARATOR() (contracts/fee-collector/interfaces/IPangolinPair.sol#18) is not in mixedCase
Function IPangolinPair.PERMIT_TYPEHASH() (contracts/fee-collector/interfaces/IPangolinPair.sol#19) is not in mixedCase
Function IPangolinPair.MINIMUM_LIQUIDITY() (contracts/fee-collector/interfaces/IPangolinPair.sol#36) is not in mixedCase
Function IPangolinRouter.WAVAX() (contracts/fee-collector/interfaces/IPangolinRouter.sol#5) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable FeeCollector.WRAPPED_TOKEN (contracts/fee-collector/FeeCollector.sol#22) is too similar to FeeCollector.constructor(address,address,address,address,address,uint256,address,address,address)._wrappedToken (contracts/fee-collector/
FeeCollector.sol#45)
Variable FeeCollector._convertLiquidity(IPangolinPair[],address).tokenOPulled (contracts/fee-collector/FeeCollector.sol#197) is too similar to FeeCollector._convertLiquidity(IPangolinPair[],address).tokenlPulled (contracts/fee-collector/
FeeCollector.sol#197)
Variable IMiniChef.addPool(uint256,address,address)._allocPoint (contracts/fee-collector/interfaces/IMiniChef.sol#29) is too similar to IMiniChef.setPools(uint256[],uint256[],address[],bool[]).allocPoints (contracts/fee-collector/interfa
ces/IMiniChef.sol#27)
Variable IMiniChef.setPool(uint256,uint256,address,bool)._allocPoint (contracts/fee-collector/interfaces/IMiniChef.sol#26) is too similar to IMiniChef.setPools(uint256[],uint256[],address[],bool[]).allocPoints (contracts/fee-collector/in
terfaces/IMiniChef.sol#27)
Variable IPangolinRouter.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (contracts/fee-collector/interfaces/IPangolinRouter.sol#10) is too similar to IPangolinRouter.addLiquidity(address,addr
ess,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (contracts/fee-collector/interfaces/IPangolinRouter.sol#11)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#130-132)
revokeRole(bytes32,address) should be declared external:
        - AccessControl.revokeRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#143-145)
renounceRole(bytes32,address) should be declared external:


        - AccessControl.renounceRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControl.sol#161-165)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

- No major issues found by Slither.

3.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits 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 all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

FeeCollector.sol

Report for contracts/fee-collector/FeeCollector.sol
https://dashboard.mythx.io/#/console/analyses/ba4c317e-51ae-44b4-af87-9130a7328089

Line	SWC Title	Severity	Short Description
90	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
107	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
162	(SWC-110) Assert Violation	Unknown	Out of bounds array access
163	(SWC-110) Assert Violation	Unknown	Out of bounds array access
166	(SWC-110) Assert Violation	Unknown	Out of bounds array access
167	(SWC-110) Assert Violation	Unknown	Out of bounds array access
168	(SWC-110) Assert Violation	Unknown	Out of bounds array access
190	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
191	(SWC-110) Assert Violation	Unknown	Out of bounds array access
214	(SWC-115) Authorization through tx.origin	Low	Use of "tx.origin" as a part of authorization control.
231	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
231	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
232	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
232	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
233	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered

- Integer Overflows and Underflows flagged by MythX are false positives, as the contract is using Solidity 0.8.9 version. After the Solidity version 0.8.0 Arithmetic operations revert to underflow and overflow by default.
- Assert violations are false positives.
- `tx.origin` is used to avoid calling the `harvest()` function from a smart contract.

THANK YOU FOR CHOOSING

// HALBORN