Pangolin - Fee Collector

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 9th, 2021 - December 11th, 2021**

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

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 December 9th, 2021 and ending on December 11th, 2021. The security assessment was scoped to the smart contract provided in the Github repository pangolindex/fee-collector.

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 identified some security risks that should be addressed by Pangolin team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of 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. 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

1.4 SCOPE

IN-SCOPE:

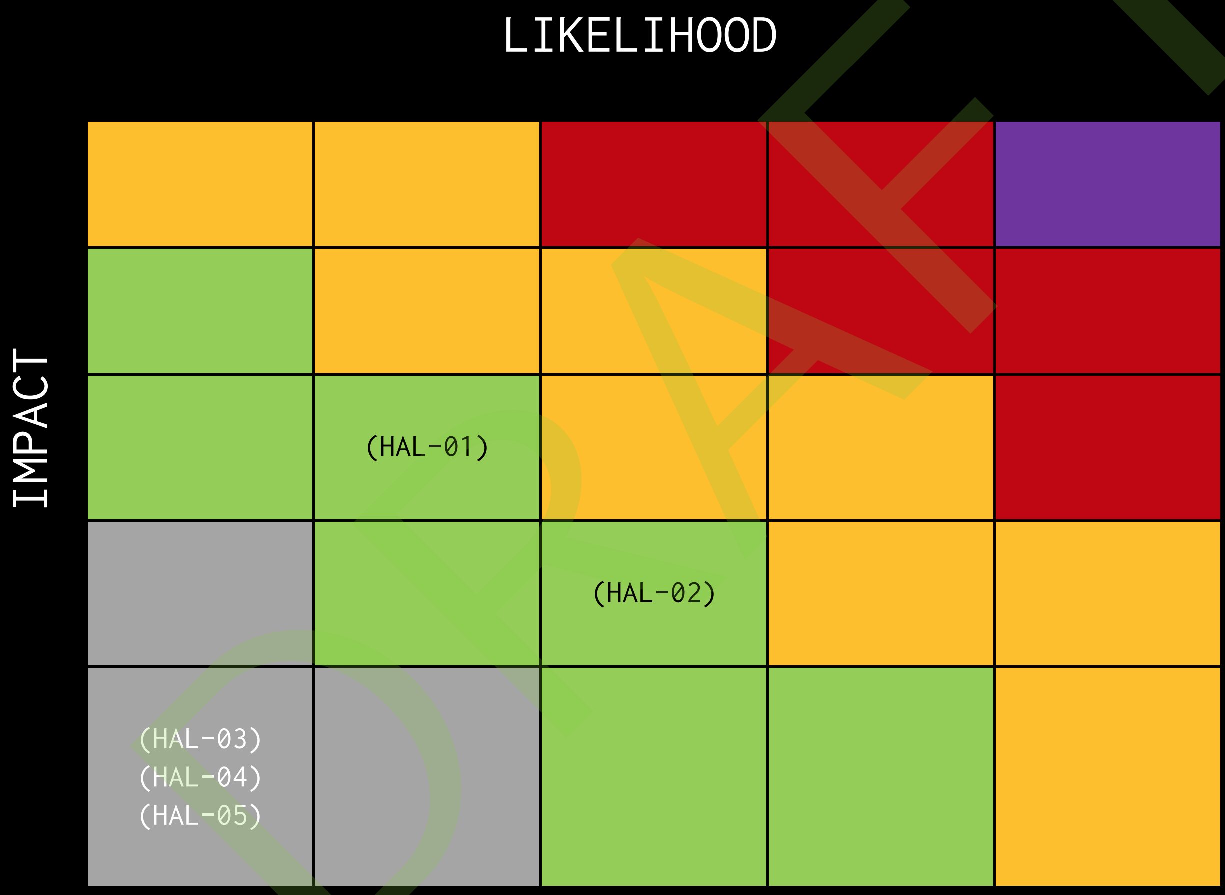
The security assessment was scoped to the following smart contracts:

- FeeCollector.sol

Commit ID: 903abab3be7af7c4266c26117d83a17a9b2922e7

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - DOS WITH BLOCK GAS LIMIT	Low	-
HAL02 - LACK OF ZERO ADDRESS CHECK	Low	-
HAL03 - USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS	Informational	-
HAL04 - NO NEED TO INITIALIZE UINT256 I VARIABLE TO 0	Informational	-
HAL05 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) DOS WITH BLOCK GAS LIMIT - LOW

Description:

When smart contracts are deployed or functions inside them are called, the execution of these actions always require a certain amount of gas, based on how much computation is needed to complete them. The Ethereum network specifies a block gas limit and the sum of all transactions included in a block cannot exceed the threshold. Programming patterns that are harmless in centralized applications can lead to Denial of Service conditions in smart contracts when the cost of executing a function exceeds the block gas limit. In the contract FeeCollector.sol, the function _collectFees() iterates over an array of liquidityPairs of unknown size. If this array is big enough, the transaction could reach the block gas limit and would not be completed.

Code Location:

Listing 1: FeeCollector.sol (Lines 123)

```
120 function _collectFees(address[] memory liquidityPairs,
121     address outputToken) internal {
122     require(outputToken != address(0), "Output token unspecified")
            ;
123     for (uint256 i = 0; i < liquidityPairs.length; i++) {
124         address currentPairAddress = liquidityPairs[i];
125         IPangolinPair currentPair = IPangolinPair(
                currentPairAddress);
126         uint256 pglBalance = currentPair.balanceOf(address(this));
127         if (pglBalance > 0) {
128             _pullLiquidity(currentPair, pglBalance);
129             address token0 = currentPair.token0();
130             address token1 = currentPair.token1();
131             if (token0 != outputToken) {
132                 _swap(token0, outputToken,
133                     IERC20(token0).balanceOf(address(this)));
134             }
135             if (token1 != outputToken) {
```

```
                    _swap(token1, outputToken,
                        IERC20(token1).balanceOf(address(this)));
                }
            }
        }
}
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to limit the size of the liquidityPairs parameter in the harvest() function. For example:

Listing 2: FeeCollector.sol (Lines 150)

```
148 function harvest(address[] memory liquidityPairs, bool
        claimMiniChef)
149 public {
150     require (liquidityPairs.length <= 30, "liquidityPairs should
            be <= 30")
151     address _outputToken = IStakingRewards(stakingRewards).
            rewardsToken();
152
153     if (claimMiniChef) {
154         IMiniChef(MINICHEF).harvest(miniChefPoolId, address(this))
                ;
155     }
156
157     if (liquidityPairs.length > 0) {
158         _collectFees(liquidityPairs, _outputToken);
159     }
160
161     uint256 _finalBalance = IERC20(_outputToken).balanceOf(address
            (this));
162
163     uint256 _callIncentive = _finalBalance * harvestIncentive
164     / FEE_DENOMINATOR;
```

```
    uint256 _totalRewards = _finalBalance - _callIncentive;

    if (_callIncentive > 0 && _totalRewards > 0) {
        IERC20(_outputToken).safeTransfer(stakingRewards,
            _totalRewards);
        // No need for event as notifyRewardAmount will create one
        IStakingRewards(stakingRewards).notifyRewardAmount(
            _totalRewards);
        IERC20(_outputToken).safeTransfer(msg.sender,
            _callIncentive);
    }

}
```

3.2 (HAL-02) LACK OF ZERO ADDRESS CHECK - LOW

Description:

The function transferStakingOwnership() is not checked against address (0). Every address should be validated and checked that is different from zero.

Code Location examples:

Listing 3: FeeCollector.sol

```
function transferStakingOwnership(address _newOwner) external
    onlyOwner {
    IStakingRewards(stakingRewards).transferOwnership(_newOwner);
}
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Validate that every address input is different from zero.

3.3 (HAL-03) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL

Description:

In the loop, the variable i is incremented using i++. It is known that, in loops, using ++i costs less gas per iteration than i++.

Code Location:

FeeCollector.sol

Line 123: for (uint256 i = 0; i < liquidityPairs.length; i++)

Proof of Concept:

For example, based in the following test contract:

Listing 4: Test.sol

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.9;

contract test {
    function postiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; i++) {
        }
    }
    function preiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; ++i) {
        }
    }
}
```

We can see the difference in the gas costs:

```
>>> test_contract.postiincrement(1)
Transaction sent: 0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 44
  test.postiincrement confirmed   Block: 13622335   Gas used: 21620 (0.32%)

<Transaction '0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b'>
>>> test_contract.preiincrement(1)
Transaction sent: 0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 45
  test.preiincrement confirmed   Block: 13622336   Gas used: 21593 (0.32%)

<Transaction '0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a'>
>>> test_contract.postiincrement(10)
Transaction sent: 0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 46
  test.postiincrement confirmed   Block: 13622337   Gas used: 22673 (0.34%)

<Transaction '0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033'>
>>> test_contract.preiincrement(10)
Transaction sent: 0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 47
  test.preiincrement confirmed   Block: 13622338   Gas used: 22601 (0.34%)

<Transaction '0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05'>
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use ++i instead of i++ to increment the value of an uint variable inside a loop. This is not applicable outside of loops.

3.4 (HAL-04) NO NEED TO INITIALIZE UINT256 I VARIABLE TO 0 - INFORMATIONAL

Description:

As `miniChefPoolId` is an `uint256`, it is already initialized to 0. `uint256 public miniChefPoolId = 0;` reassigns the 0 to `miniChefPoolId` which wastes gas. The same applies to the `uint256 i` variable declared in the loop.

Code Location:

FeeCollector.sol
Line 27: `uint256 public miniChefPoolId = 0;`
Line 123: `for (uint256 i = 0; i < liquidityPairs.length; i++)`

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not initialize the two variables mentioned to 0 to save some gas. For example:
Line 27: `uint256 public miniChefPoolId;`
Line 123: `for (uint256 i; i < liquidityPairs.length; i++)`

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

Description:

In the PangolinFeeCollector contract there is a function marked as public but it is never directly called within the same contract or in any of their descendants:

FeeCollector.sol
- harvest() (FeeCollector.sol#148-174)

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If the function is not intended to be called internally or by their descendants, it is better to mark it as external to reduce gas costs.

AUTOMATED TESTING

4.1 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

```
PangolinFeeCollector._pullLiquidity(IPangolinPair,uint256) (contracts/FeeCollector.sol#71-82) ignores return value by pair.approve(PANGOLIN_ROUTER,balance) (contracts/FeeCollector.sol#72)
PangolinFeeCollector._pullLiquidity(IPangolinPair,uint256) (contracts/FeeCollector.sol#71-82) ignores return value by IPangolinRouter(PANGOLIN_ROUTER).removeLiquidity(pair.token0(),pair.token1(),balance,0,0,address(this),block.timestamp
+ 60) (contracts/FeeCollector.sol#73-81)
PangolinFeeCollector._swap(address,address,uint256) (contracts/FeeCollector.sol#89-114) ignores return value by IPangolinRouter(PANGOLIN_ROUTER).swapExactTokensForTokens(amount,0,path,address(this),block.timestamp + 60) (contracts/FeeCol
lector.sol#107-113)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

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

PangolinFeeCollector.setHarvestIncentive(uint256) (contracts/FeeCollector.sol#45-48) should emit an event for:
        - harvestIncentive = _harvestIncentive (contracts/FeeCollector.sol#47)
PangolinFeeCollector.setMiniChefPool(uint256) (contracts/FeeCollector.sol#52-54) should emit an event for:
        - miniChefPoolId = _pid (contracts/FeeCollector.sol#53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

PangolinFeeCollector._collectFees(address[],address) (contracts/FeeCollector.sol#120-141) has external calls inside a loop: pglBalance = currentPair.balanceOf(address(this)) (contracts/FeeCollector.sol#126)
PangolinFeeCollector._collectFees(address[],address) (contracts/FeeCollector.sol#120-141) has external calls inside a loop: token0 = currentPair.token0() (contracts/FeeCollector.sol#129)
PangolinFeeCollector._collectFees(address[],address) (contracts/FeeCollector.sol#120-141) has external calls inside a loop: token1 = currentPair.token1() (contracts/FeeCollector.sol#130)
PangolinFeeCollector._collectFees(address[],address) (contracts/FeeCollector.sol#120-141) has external calls inside a loop: _swap(token0,outputToken,IERC20(token0).balanceOf(address(this))) (contracts/FeeCollector.sol#132-133)
PangolinFeeCollector._collectFees(address[],address) (contracts/FeeCollector.sol#120-141) has external calls inside a loop: _swap(token1,outputToken,IERC20(token1).balanceOf(address(this))) (contracts/FeeCollector.sol#136-137)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#26-36) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#32-34)
Address._verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#189-209) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#201-204)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
        - Version used: ['0.8.9', '^0.8.0']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3)
        - 0.8.9 (contracts/FeeCollector.sol#1)
        - 0.8.9 (interfaces/IMiniChef.sol#2)
        - 0.8.9 (interfaces/IStakingRewards.sol#1)
        - 0.8.9 (interfaces/Pangolin/IPangolinPair.sol#2)
        - 0.8.9 (interfaces/Pangolin/IPangolinRouter.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#79-81) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#108-114) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#168-170) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#178-187) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#141-143) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#151-160) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#54-59) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#20-22) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#68-79) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#59-66) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#28-35) 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/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (contracts/FeeCollector.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IMiniChef.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IStakingRewards.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/Pangolin/IPangolinPair.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/Pangolin/IPangolinRouter.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
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#54-59):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#122-133):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#131)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#151-160):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#158)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#178-187):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

AUTOMATED TESTING

```
Parameter PangolinFeeCollector.setRewardsContract(address)._stakingRewards (contracts/FeeCollector.sol#38) is not in mixedCase
Parameter PangolinFeeCollector.setHarvestIncentive(uint256)._harvestIncentive (contracts/FeeCollector.sol#45) is not in mixedCase
Parameter PangolinFeeCollector.setMiniChefPool(uint256)._pid (contracts/FeeCollector.sol#52) is not in mixedCase
Parameter PangolinFeeCollector.setRewardsDuration(uint256)._rewardsDuration (contracts/FeeCollector.sol#58) is not in mixedCase
Parameter PangolinFeeCollector.transferStakingOwnership(address)._newOwner (contracts/FeeCollector.sol#64) is not in mixedCase
Function IPangolinPair.DOMAIN_SEPARATOR() (interfaces/Pangolin/IPangolinPair.sol#19) is not in mixedCase
Function IPangolinPair.PERMIT_TYPEHASH() (interfaces/Pangolin/IPangolinPair.sol#20) is not in mixedCase
Function IPangolinPair.MINIMUM_LIQUIDITY() (interfaces/Pangolin/IPangolinPair.sol#37) is not in mixedCase
Function IPangolinRouter.WAVAX() (interfaces/Pangolin/IPangolinRouter.sol#6) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable IMiniChef.setPool(uint256,uint256,address,bool)._allocPoint (interfaces/IMiniChef.sol#26) is too similar to IMiniChef.setPools(uint256[],uint256[],address[],bool[]).allocPoints (interfaces/IMiniChef.sol#27)
Variable IMiniChef.addPool(uint256,address,address)._allocPoint (interfaces/IMiniChef.sol#29) is too similar to IMiniChef.setPools(uint256[],uint256[],address[],bool[]).allocPoints (interfaces/IMiniChef.sol#27)
Variable IPangolinRouter.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (interfaces/Pangolin/IPangolinRouter.sol#11) is too similar to IPangolinRouter.addLiquidity(address,address,uint256,uin
t256,uint256,uint256,address,uint256).amountBDesired (interfaces/Pangolin/IPangolinRouter.sol#12)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#53-55)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#61-64)
harvest(address[],bool) should be declared external:
        - PangolinFeeCollector.harvest(address[],bool) (contracts/FeeCollector.sol#148-174)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

- No major issues found by Slither.

4.2 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:

MythX only found some issues in the following smart contracts:

FeeCollector.sol

Report for contracts/FeeCollector.sol
https://dashboard.mythx.io/#/console/analyses/9ea7ceba-9b68-4d36-bfae-4465ced716c7

Line	SWC Title	Severity	Short Description
80	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
96	(SWC-110) Assert Violation	Unknown	Out of bounds array access
97	(SWC-110) Assert Violation	Unknown	Out of bounds array access
100	(SWC-110) Assert Violation	Unknown	Out of bounds array access
101	(SWC-110) Assert Violation	Unknown	Out of bounds array access
102	(SWC-110) Assert Violation	Unknown	Out of bounds array access
112	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
123	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
124	(SWC-110) Assert Violation	Unknown	Out of bounds array access
163	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
163	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
165	(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.

THANK YOU FOR CHOOSING

// HALBORN